UNIVERSITY OF OSLO
Department of Informatics

Dynamic benchmarking in bioinformatics

Master thesis

Anders Ramsvik Bragstad

May 1, 2013

Acknowledgments

First of all I want to thank my advisors Torbjørn Rognes and Geir Kjetil Sandve for coming up with a fun and interesting topic to write about. And thank you for your great insight about the topic and for good advice underway.

Next I want to give a special thanks to all the friendly and helpful people on the 10th floor. Especially, I want to thank Kai and Sveinung for giving great answers and helpful guidance to all my technical questions and troubles, and Nils for excellent feedback and advice throughout the writing process.

Finally I want to give a big thanks to my family for their care and support during the entire master process, and especially in the last stressful weeks.

Abstract

In bioinformatics there often exist a wide selection of different algorithms to solve a given problem. Therefore, benchmarks are often used by researchers to either find the most suitable algorithms for their specific need, or compare their newly developed algorithm to existing ones.

Though benchmarks can be useful for finding the right algorithm for the problem, good and relevant benchmarks are often lacking in many areas. This is due to the fact that benchmarks tend to be quite difficult and time consuming to develop, and it's hard to keep up to date with new developments.

We here present an online benchmarking system, integrated in a Galaxy-based platform. This is a system aimed at biologists with little or no programming experience as well as bioinformaticians who don't want to build benchmark functionality from scratch. The purpose is to provide a tool that enables users to easily create and share benchmarks and evaluate test sets that reflects the users specific needs. The tool streamlines each step of the benchmarking process, and supports benchmarking for a selection of different problem areas such as gene prediction, motif discovery, splice site prediction and nucleosome prediction.

The system was evaluated with six different test sets, assessing the systems capability of handling benchmarks across different problem areas, sizes and types. Despite some current limitations, the system shows promise as a potential platform for benchmarking in bioinformatics.

The system is available for use at http://hyperbrowser.uio.no/bench/, and consists of three tools; Benchmark Creation Tool, Benchmark Retrieval Tool and Benchmark Evaluation Tool.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Motivation

The field of bioinformatics is a huge area of research that gives rise to several different problem areas. Each of these problem areas tend to have a high grade of complexity, and can be considered extremely difficult to solve. A lot of effort has been put into creating sophisticated algorithms to solve these problems, and a wide selection of software is available for each given problem with varying quality and strong/weak points.

Therefore benchmarking can be a useful tool for evaluating the results of a selection of different algorithms, in order to assess which algorithm gives the best results. Though useful, good benchmarks are often lacking in many areas, making good evaluations of algorithms difficult.

In this thesis we will look into ways to improve how benchmarks are carried out, from creation to evaluation. Then we will present a benchmark system that addresses some of the problems of today's benchmarks. Finally we'll test our system using real test sets of different kind, size and across different problem areas.

With this system we want to find answers to the following research questions:

1. Is it possible to streamline the approach of creating and evaluating benchmarks for similar problem areas?

2. Can this streamlined approach be flexible in both size and type of benchmark, to fit the needs of the user?

3. Can the same approach be expanded to fit other problem areas that requires different evaluation?

Chapter 2

Background

In this chapter we'll first give a short introduction of the biological background. Next, we'll give an introduction to benchmarking and how it's used in bioinformatics. Then, we'll take a look at some of the problem areas which will be relevant to benchmark later on. Next, we'll look into some of the common evaluation methods for these kinds of problems. And finally we will look into some relevant technologies for creating a benchmarking system.

2.1 Biological background

Here we will give an introduction to some relevant biology for this project. Many details are left out, but it should be enough to understand the biological foundation for the problem areas we are going to look into later on.

2.1.1 DNA

Every living cell in both plants, animals and microorganisms contain genetic information, which is stored in long strands of *Deoxyribonucleic acid* or *DNA*.

DNA is made out of four chemical components: *adenine* (A), *guanine* (G), *cytosine* (C) and *thymine* (T). These components, called *nucleotides*, are formed into pairs, called *base pairs*. Adenine always pairs with thymine, and cytosine always pairs with guanine. This way each nucleotide complements the other, making the strands anti parallel. Together these base pairs form long DNA sequences called *chromosomes*, which curls around in a spiral, known as the *double helix*.

Having a double anti parallel strand makes DNA able to replicate genetic content, given only information from one strand. This is important for DNA replication to work, where chromosomes are copied into new chromosomes. These new chromosomes will then be provided for a new cell to be formed during cell division [26].

As we can see from Figure 2.1 the DNA strands run in opposite directions, one sequence containing the nucleotides GACTTGATACTTG,

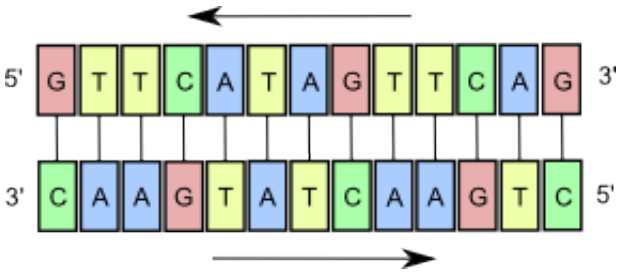

Figure 2.1: A set of base pairs, forming a DNA strand.

and the other containing the nucleotides CAAGTATCAAGTC.

2.1.2 Genes

Roughly 1.5 % of the DNA consists of *genetic regions*, which holds the information for all the biological traits of the given species. The genetic regions contains codes for various *amino acids*, which are the building blocks for proteins [26].

2.1.3 Transcription

The whole process of creating proteins out of protein-coding genes starts of with a process called *transcription*. This process is initiated by a protein called the *transcription factor* binding with a part of the DNA called the *promoter region*. The promoter region is located nearby the start of a gene.

The transcription factor can then recruit an enzyme called the *RNA polymerase*. After the RNA polymerase makes contact with the transcription factor, the RNA polymerase runs across the entire gene, producing a *RNA molecule* based on the genomic instructions found within the DNA sequence, until it reaches a stop sign [26].

2.1.4 RNA splicing

Before the RNA molecule can be translated into a protein, it have to go through a process called *RNA splicing*. The RNA chain consists of two kinds of sequences; *introns* and *exons*. In RNA splicing, introns are removed from the RNA chain, so that only the exons remain. It is still unclear what is the exact purpose of the introns, though there are several theories [8].

Figure 2.2: Shows the structure of a gene, from pre-mRNA to mRNA.

2.1.5 DNA packaging

DNA packaging is the process of packaging an entire DNA strand into the shape of a chromosome, which is the shape the DNA takes during cell division. This process starts off with nine different proteins, called *histones*, position themselves along the DNA strand. Together these histones form a protein called a *nucleosome*. The DNA then coils around this nucleosome, and together with several other nucleosomes are packed into something called *chromatin*. The DNA is then further packaged using other proteins to form a chromosome [3].

2.1.6 Prokaryotes and eukaryotes

All organisms can be categorized as either *prokaryotes* or *eukaryotes* depending on their cell structure. The main difference between the two is that prokaryotic cells do not have a cell nucleus, while eukaryotic cells do.

Eukaryotes are complex beings such as plants, insects and animals, whereas prokaryotes are simple organisms like bacteria [35].

2.2 Benchmarking in bioinformatics

In this section we will give a brief introduction to benchmarking. Then we will look at some popular benchmarks which are used in bioinformatics today.

2.2.1 Benchmark basics

Even though there are differences in the problems we benchmark, they all share the same workflow and the same components as outlined in figure 2.3.

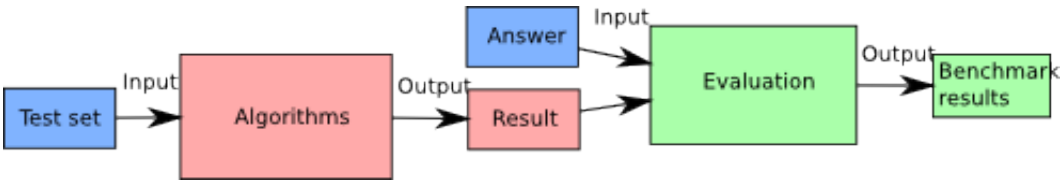

Figure 2.3: The basic workflow of every benchmark.

The blue components are created by the *benchmark creators*, which are the *test set* and the *answer*. The test set is given as input to the algorithms. Often several test sets are defined to find different strong and weak points of an algorithm. This is called a *benchmark suite*. The answer contains the correct results of the test sets.

The red components consist of the *algorithm* and the *result*. The algorithm can be any software solution developed for this problem area, and normally the test set is run on several algorithms to give a good

comparison. The result is the output that the algorithm returned with the given test set.

The green components are created by the *benchmark developer*, which are the *evaluation* and *benchmark results*. The evaluation is the software which compares the answer with the result from the algorithm, and outputs a benchmark result which shows us how well the result matched the answer.

We want to distinguish between the benchmark creator (the one who creates the test sets) and the benchmark developer (the one who develops the evaluation), though these parts are often developed by the same person. In this thesis, however, we will focus on evaluation development and leave the test set creation up to others.

2.2.2 Existing benchmarks

Now that we have looked at what a benchmark is, let's have a look at some specific benchmarks. The list of benchmarks in bioinformatics is long, but we can take a closer look at two of the more popular benchmarks used in bioinformatics today:

- **Critical Assessment of Techniques for Protein Structure Prediction (CASP)**: CASP is a large scale experiment which has been carried out every two year since 1994. Its goal is to assess different structure prediction methods for proteins, provided by various research groups [25]. This experiment can also be considered a competition, where the research groups are competing against each other to come up with the best results. Therefore there are new test sets available every two years. In 2010 there were 129 test sets, which consists of amino acids sequences for different proteins.

- **BALIBASE**: BALIBASE is an open source benchmarking system written in C, and is the most widely used benchmarking system for multiple sequence alignment (the process of aligning multiple sequence to find a common evolutionary origin). The newest version (3.0) contains 6255 sequences and 217 alignments based on 3D structure supersitions. These are separated into nine different reference sets which are both complex and challenging, and are designed to represent many of the problems encountered by multiple alignment methods today [39].

2.3 Problem areas

In this section we'll introduce the problem areas which will be relevant to benchmark later on. We will mostly focus on motif discovery and gene prediction in this thesis, though the same approach can also be used in other problem areas sharing the same input and output such as splice site prediction and nucleosome prediction.

2.3.1 Gene prediction

One major challenge in bioinformatics is locating the regions of the DNA that encode genes. The problem of *gene prediction* consists of predicting the location of protein coding regions and the functional sites of genes, given a DNA sequence.

The most common way of finding the location of genes is with *Ab Initio* gene prediction. This method relies on two types of sequence information. The first is signal sensors which can be short sequence motifs, such as splice sites, branch points, polypyrimidine tracts, start codons and stop codons. The other is content sensors that uses statistics detection algorithms to distinguish species specific patterns within the sequences [41].

Gene prediction can be further categorized depending on the organism being a prokaryot or eukaryot. Gene prediction for prokaryotes is a well studied and, some would say solved problem. Gene prediction for eukaryotes is more complex, and difficult to predict [22].

2.3.1.1 Existing gene prediction algorithms

There have been developed a lot of different software for gene prediction. Here we will present some of the more significant gene prediction software.

1. **Genemark:** Genemark is a gene prediction algorithm first introduced in 1993 **[6].** It was a pioneering algorithm at the time, introducing the so called "three-periodic Markov model" [24]. Now, improved versions have been introduced such as Genemark.hmm which uses a hidden markov model [23].

2. **Glimmer:** Glimmer is a gene prediction algorithm introduced in 1998. It uses interpolated Markov models (IMMs) as a framework for capturing dependencies between nearby nucleotides in a DNA sequence [31].

3. **Prodigal:** Prodigal is a gene prediction algorithm introduced in 2009. Its implementation is based on dynamic programming **[22].**

2.3.1.2 Previous work in gene prediction benchmarking

The number of articles evaluating gene prediction algorithms is quite large. Many of them however is quite outdated, and does not include new algorithms such as Prodigal. Here we will present two of the more recent evaluations written about gene prediction, one from 2005 and one from 2012:

- The article "Evaluation of five ab initio gene prediction programs for the discovery of maize genes" evaluates the accuracy of five different gene prediction algorithms on the maize genome: FGENESH, Genemark.hmm, GENSCAN, GlimmerR and Grail. The result showed that FGENESH came up with the most accurate results, followed by Genemark.hmm [44].

- The master thesis "Evaluation of gene prediction methods for prokaryotes" evaluates the accuracy of the gene prediction algorithms Genemark.hmm, GenemarkS, Glimmer, MED and Prodigal on prokaryotes. The thesis indicated that Prodigal outperformed the older algorithms [12].

2.3.2 Motif discovery

A transcription factor binding site is where the transcription factor binds along the DNA in order to initiate transcription. The problem of *motif discovery* is the process of discovering these binding sites for one or more transcription factors from a set of DNA sequences assumed to be in regulatory regions [40].

The most common approach for motif finding is to extract a set of sequence from the genome, and then search for the most overrepresented motifs according to some motif model [34].

2.3.2.1 Existing motif discovery algorithms

Just like gene prediction, there exists many algorithms for motif discovery. Here we will present some of the more popular algorithms used today:

- **MEME:** Multiple EM for Motif Elicitation or MEME is a motif discovery algorithm first introduced in 1996. It can be used to discover the binding sites for transcription factors, as well as protein domains. The algorithm search for repeated, ungapped sequence patterns in the DNA sequences **[5].**

- **Weeder:** Weeder is a motif discovery algorithm first introduced in 2001 [27]. It predicts transcription factor binding sites using an exhaustive search algorithm **[28].**

- **YMF:** Yeast Motif Finder or YMF is a motif discovery algorithm first introduced in 2000 **[36].** It uses an exhaustive search algorithm to predict transcription factor binding sites **[37].**

2.3.2.2 Previous work in motif discovery benchmarking

Like gene prediction, there have been written numerous articles evaluating the results of motif discovery programs. Here, we will present two articles about benchmarking motif discovery algorithms:

- The article "Assessing computational tools for the discovery of transcription factor binding sites" evaluates 14 different motif discovery algorithms, among them is MEME, Weeder and YMF. The result showed that Weeder outperformed the other algorithms on most data sets, though MEME and YMF performed better on the mouse data sets [40].

- The article "Improved benchmarks for computational motif discovery" describes a benchmarking suite used to benchmark motif discovery algorithms. The paper discusses how current synthetic data sets tend to have a certain bias towards specific motif models. Then, it proposes a new way of constructing data sets, which is by collecting binding site fragments that are ranked according to the optimal level of discrimination. The benchmark suite consists 228 test sets in total [33].

2.3.3 Splice site prediction

Splice site prediction is the process of discovering the splice sites where introns and exons meet within a gene [45]. When we run splice site programs we end up with a list of potential splice sites were various gene structures might be built. They are therefore useful in addition to gene prediction to refine an existing gene structure [24].

2.3.4 Nucleosome prediction

Nucleosome prediction is the process of predicting where nucleosomes position themselves along the DNA sequence, by looking for patterns in the DNA itself. Nucleosomes are the main component used in DNA packaging, which is the process of coiling the DNA strand into a chromosome [38].

2.3.5 Problem similarities and terminology

The reason we are looking at these problem areas is that although they are solving different problems, they are in fact identical when it comes to their input and output.

As input they take one or more DNA sequences, which contains strands of base pairs in the form of A, C, G and T's. The algorithm then tries to predict the locations for either genes, motifs, splice sites or nucleosome positions, from now on referred to as a common *pattern* in the sequence.

And as output, they return their *predictions* based on the input sequences. These predictions contain locations of where the algorithm have predicted there to be patterns. These are essentially a set of start and end positions along the input sequence, which from now on will be referred to as *segment*s. These segments in most cases also comes with a *score*, which tells us how the algorithm ranks this segment over other segments.

These predictions can then be evaluated by comparing the prediction segments with the answer segments, by measuring how well they overlap each other.

Because of these similarities we should in theory be able to use the same approach both for creating test sets, and evaluating benchmarks for these kinds of problems.

2.4 Evaluation methods

Giving the benchmark results a meaningful and accurate evaluation is an essential part of any benchmarking system, as this is how the results will be presented for the end user. Here, we will introduce some relevant methods for evaluating test results for the problem areas introduced in section 2.3.

We will start by presenting binary classification, a class of problems common in bioinformatics (and other fields for that matter), and discuss some of the common ways of evaluating such problems. Next, we will look at base pair probability predictions, which is another type of prediction requiring somewhat different evaluation.

2.4.1 Evaluation of binary classification problems

What all the problems we introduced in section 2.3 have in common is that they can be thought of as a *binary classification problem*. A binary classification problem is the task of dividing a set of objects into two groups based on having a specified classification property.

For example, let's take the problem area of gene prediction. In this case the classification property is a genetic region. So for every base pair along the DNA sequence, we want to divide them into two groups. One that is within a genetic region, and another one that is not.

As Table 2.1 shows, the results from a binary classification problem can end up in four different categories. Both True Positives (TP) and True Negatives (TN) are positive results where the results and answer shows the same results. Whereas False Positive (FP) and False Negatives (FN) are incorrect results, also known as type 1 and type 2 errors respectively. With type 1 errors the results comes up positive, but in reality it's negative, whereas with type 2 errors the results comes up negative, but in reality it's positive.

	Answer: True	**Answer: False**
Result: True	True Positive (TP) = 12	False Positive (FP) = 6
Result: False	False Negative (FN) = 4	True Negative (TN) = 78

Table 2.1: Shows the different possible outcomes of a binary classification problem. The numbers are example values, representing the number of instances of each category.

2.4.1.1 Statistical measurements

For a binary classification problem, we can come up with several different statistical measurements to evaluate the performance, given a set of occurrences of TP, FP, FN and TN. Here, we look at some of them which can be relevant for benchmark evaluation. We take the values from table 2.1 as an example of a test result, and table 2.2 will display the formula and the results from the test.

Name	Formula
Sensitivity	$^{TP}/_{(TP+FN)} = {}^{12}/_{(12+4)} = 0.75$
Specificity	$^{TN}/_{(TN+FP)} = {}^{78}/_{(78+6)} = 0.928$
Accuracy	$^{(TP+TN)}/_{n} = {}^{(12+78)}/_{100} = 0.9$
PPV	$^{TP}/_{(TP+FP)} = {}^{12}/_{(12+6)} = 0.667$
NPV	$^{TN}/_{(TN+FN)} = {}^{78}/_{(78+4)} = 0.951$
PC	$^{TP}/_{(TP+FN+FP)} = {}^{12}/_{(12+4+6)} = 0.545$
CC	$\frac{(TP*TN-FP*FN)}{\sqrt{((TP+FP)*(FP+TN)*(TN+FN)*(FN+TP))}} = \frac{(12*78-6*4)}{\sqrt{((12+6)*(6+78)*(78+4)*(4+12))}} = 0.648$
ASP	$^{(Sn+PPV)}/_{2} = {}^{(0.75+0.667)}/_{2} = 0.708$

Table 2.2: An overview of some useful statistical measurement used in evaluating benchmarks. Calculations are based on the values from table 2.1.

These measurements have in common that the higher score (closer to 1), the better results. Each measurement highlight different attributes of the result, which will be explained in further detail below:

- **Sensitivity (Sn):** Also called TP rate or recall. This measurement tell us that given a positive answer, what are the probability the prediction will also give a positive result.
- **Specificity (Sp):** Also called FP rate. This measurement tell us that given a negative answer, what are the probability the prediction will produce a negative result.
- **Accuracy:** Shows the overall accuracy of the prediction. Meaning given any result, what is the probability its correct.
- **Positive Predictive Value (PPV):** Also known as precision. This measurement gives us the probability that a positive prediction is correct.
- **Negative Predictive Value (NPV):** This measurement gives us the probability that a negative prediction is correct.
- **Performance Coefficient (PC):** Captures both properties of Sensitivity and Positive Predictive Value in one measurement.
- **Correlation Coefficient (CC):** This measure shows the overall quality of a result, where 1 means a perfect prediction, 0 a total random prediction, and -1 means a total opposite of the answer. CC can therefore be the best measurement to look at for determining the overall quality of the algorithm.
- **Average Site Performance:** A more direct approach way to combine the sensitivity and Positive Predictive Value, by combining the two measurements and dividing them by two [11].

These measurements can together give us a good idea about the overall performance with measurements like the correlation coefficient on one

hand. On the other hand, the sensitivity and specificity values can together give us an idea of the strictness of the prediction.

A high sensitivity, low specificity result would indicate a result that comes up with a positive result too often, and one should consider making the prediction stricter. On the contrary, a high specificity, low sensitivity would mean a result that comes up with too many negative results, and would indicate a prediction which is too strict. To have accurate result, one would therefore need both high sensitivity and specificity. We will look further into the correlation between sensitivity and specificity in the next section about ROC curves.

2.4.1.2 Receiver operating characteristic

A *receiver operating characteristic curve* (or simply *ROC curve*), is a graphical plot which displays the results of a binary classification problem. Here we put one minus the specificity along the X axis, and sensitivity along the Y axis.

A result with low sensitivity and high specificity would therefore end up in the bottom left corner, while a result with high sensitivity and low specificity would end up in the top right corner. As we want results with a high sensitivity and specificity, a good result would therefore end up in the top left corner in ROC space.

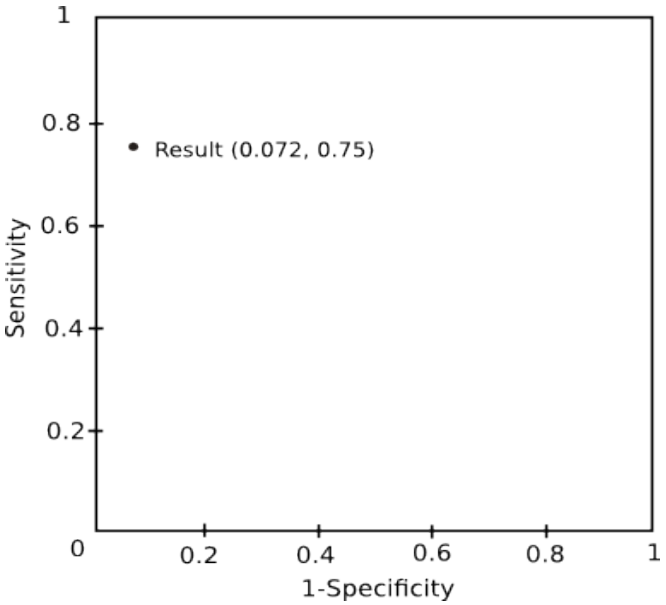

Figure 2.4: The sensitivity and specificity values from table 2.2 plotted in ROC space.

When we are evaluating predictions, we are often not interested in just plotting one point in ROC space however. Many times a binary classification problem ranks every positive instance with a score value, as we introduced in section 2.3.5. The higher the score, the higher the algorithm ranks this instance over other instances. In these cases we want

to see the performance of the results with a varying degree of sensitivity and specificity to create a ROC curve.

To create a ROC curve we first sort the instances from highest to lowest score, and then plot the instances with the highest score first. If the instance is True, the curve will move up along the Y axis and if the instance is False the curve moves straight along the X axis. By plotting these values in ROC space we end up with a curve shape as shown in 2.5.

Once we have a ROC curve we can evaluate the result by calculating the area under the curve. This way we can get a measurement of the ROC curve as a single digit between 1 and 0, where 1 means a perfect score, and 0.5 a completely random score (ie. a curve that goes right through the middle of ROC space).

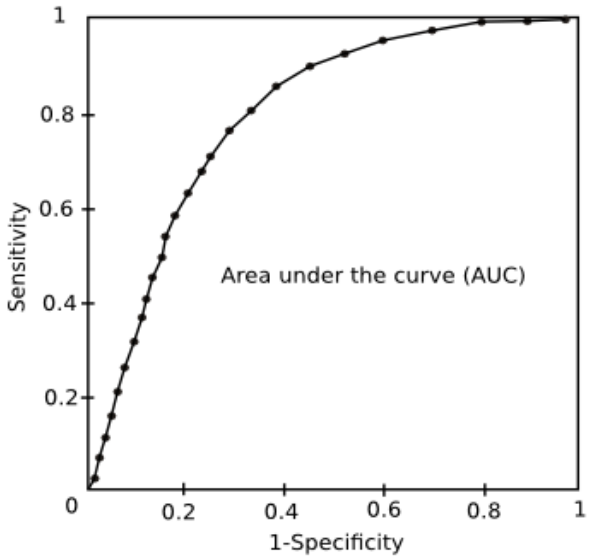

Figure 2.5: An example of a ROC curve, not based on actual data.

The area under the curve (AUC) value is an important statistical measurement, as it tells us the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance [13].

2.4.1.3 Precision recall curves

Even though ROC curves give us a good idea about the overall quality of a prediction, they are also observed to come up with overly optimistic results in the cases of imbalanced data sets. Imbalanced data in bioinformatics is a common problem, which basically boils down to test sets which has a huge over-representation of negative over positive instances. *Precision recall curves* have shown to provide a more accurate evaluation in the case of imbalanced data sets [19].

With PR curves we plot the precision (positive predictive value) along the Y axis, and the recall (sensitivity) along the X axis as displayed in figure

2.6.

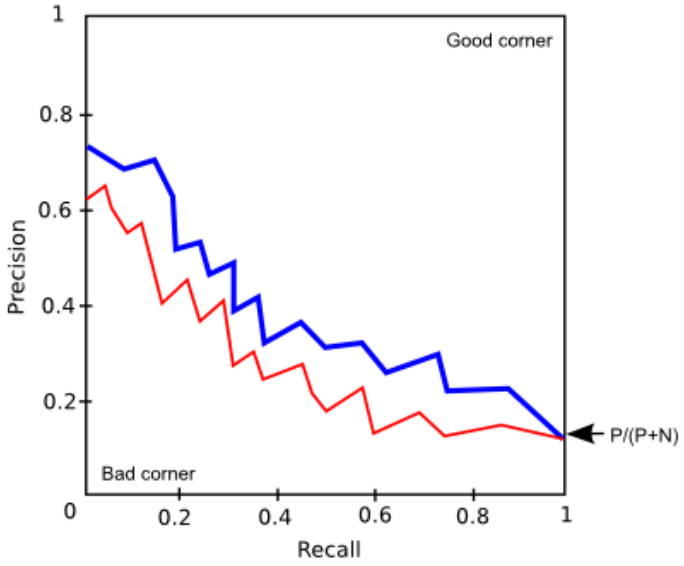

Figure 2.6: An example of PR curves, not based on actual data.

There is a connection between ROC curves and PR curves. That is a curve in ROC space dominates another curve if and only if it also dominates in PR space [9]. From the figure we can see that the blue curve dominates the red as it has a higher value without ever crossing the red curve.

2.4.2 Evaluation of base pair probability predictions

Another problem we will look into is the evaluation of *base pair probability predictions*. In this case the algorithm scores each base pair with a value between say 0 and 1. This way the values together form a function as shown in figure 2.7.

This approach is similar to many algorithms we've introduced earlier, which uses scores to rank their predictions. The difference is that these algorithms scores each predicted segment, but does not score each single base pair. The method of scoring each base pair can be applied to the same problems described in section 2.3, but will require somewhat different evaluation as there are no TP, FP, FN and TN involved.

Base pair probability predictions is a new type of prediction, based on a case of another master student working with developing machine learning algorithms to find bindings sites in chromatin states[18], a form of motif discovery. The data sets used in this thesis has a huge search space of tens or hundreds of million base pair, with only a handful of positive (normally less than 50), making the data sets extremely imbalanced.

Papers written on the evaluation of function tracks is quite absent in comparison to the evaluation of binary classification problems, as this is a quite new and different way of predicting motifs in a data set. But still

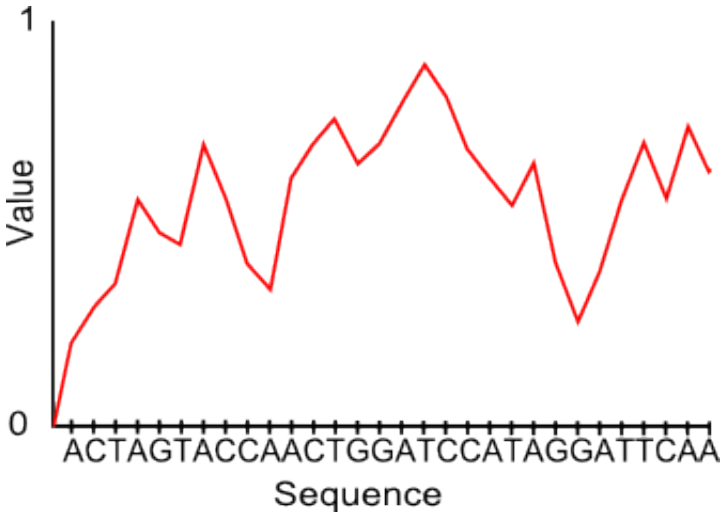

Figure 2.7: An algorithm scores each base pair, forming a function.

there are some methods we can use, which of two of them have already been introduced:

- **ROC curves:** Was introduced in section 2.4.1.2, and is one type of evaluation which can be applied to this problem. In this case though, it scores each base pair along the sequence instead of each segment.
- **PR curves:** Was introduced in section 2.4.1.3, and is another plot which can be applied to this problem. This evaluation might be particular useful here because of the imbalanced test sets used in this benchmark.

2.4.2.1 Difference comparison statistic

One simple approach for evaluating base pair probability predictions is to gather the average score within answer segments and compare them with the average score outside answer segments, and look at the difference. A high difference between the two would mean the algorithm scores the answer segments higher than non answer segments.

A difference of 0.0 means that there is no difference between the values of the function within and outside answer segments, and the higher the difference is the better the algorithm is. As we can see from figure 2.8, the blue algorithm clearly dominates the red.

2.4.2.2 Score distribution curves

A final evaluation for base pair probability predictions is to plot the *score distributions* of the function, and compare it with a plot of the score distributions of the answer. Figure 2.9 shows an example of what a score distribution curve might look like, with one red function which shows the score distribution of the entire function, and a green function which shows the distribution of scores within the answer.

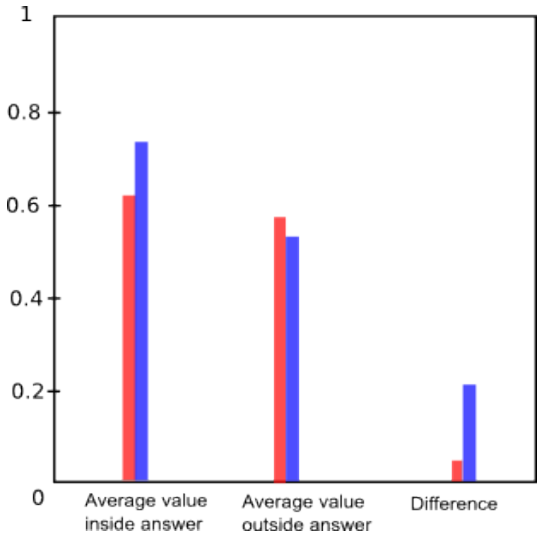

Figure 2.8: Displays the difference between scores inside and outside answer segment, not based on actual data.

As we can see a good result will have the green curve more to right of the red curve, which indicates the answer segments received higher than average scores.

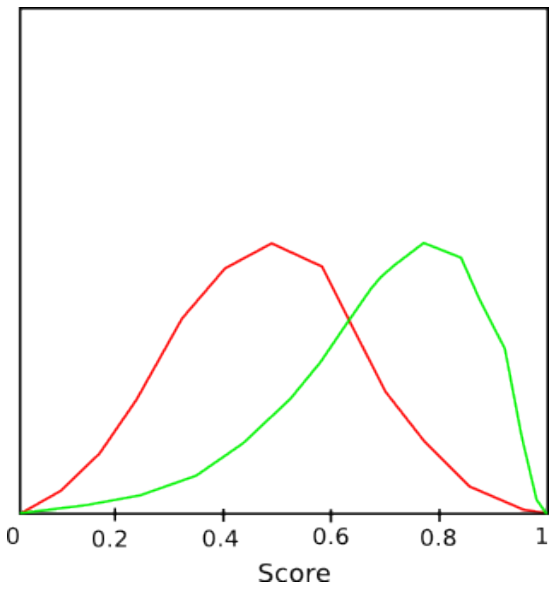

Figure 2.9: A score distribution curve based on arbitrary data.

2.5 Technologies

In this section we will introduce some relevant technologies for implementing the benchmarking system.

2.5.1 Python

Python is a dynamic programming language, meaning it requires no compilation, but executes at runtime. Python code is known to be compact and easy to read, and has a bit of a pseudo code feel to it. Even though Python is mainly a scripting language it also support object orientation, which also makes it a viable choice for larger applications [14].

Python also offers a wide selection of useful libraries, and we will now present some libraries which will be relevant to our application.

2.5.1.1 Numpy

One of the biggest complaints about the python programming language is that the performance is not up to par with other programming languages like C++ and Java. And in fields like bioinformatics where we do complex computations of huge amounts of data, speed is essential. Luckily there exists libraries in Python that can speed things up.

Numpy is an extension of python that offers computations of N-dimensional arrays written in C and Fortran [10]. Using Numpy we can do pretty much any computation imaginable on big data sets with very little code, and the execution time will be comparable with a program written in C.

2.5.1.2 Python regular expressions

As Python is commonly used as a scripting language, one of its main application is text processing and extracting useful information from text documents. Therefore Python comes with support for *regular expressions*, which is a powerful language for extracting parts of a texts [15].

2.5.1.3 Matplotlib

Matplotlib is a popular 2D graphics package for creating graphical plots in python [21]. This package can represent almost any plot possible, and is widely used by scientists worldwide.

2.5.2 Galaxy

Galaxy is an open web-based platform for genomic research, written in Python. The purpose of Galaxy is essentially to provide a set of tools which can be used to perform different computational analysis of genomic data [16].

2.5.3 Hyperbrowser

The *Genomic Hyperbrowser* is a stand-alone web application written in Python, which is tightly connected to the Galaxy framework. This means Hyperbrowser still have offers the same tools and functionality as Galaxy,

in addition to some functionality of its own. Hyperbrowser is open source as well, and encourages extension of new tools from the community [32].

2.5.4 GTrack

GTrack is a tabular file format to represent genomic tracks. A *track* is essentially a collection of information associated to specified positions along a genome.

Its purpose is to provide a format that offers flexible representations of genomic features, precise interpretation, simple parsing as well as easy conversion to many existing file formats [17].

2.5.5 FASTA format

The *FASTA format* is a common format used in bioinformatics for representing DNA and protein sequences, and is used as input for all the algorithms we've been looking at in section 2.3. FASTA was originally a protein sequence alignment program [29], which used the FASTA format as its input.

2.5.6 BLAST

Basic Local Alignment Search Tool or BLAST is a sequence alignment algorithm. This tool takes a FASTA file as input, and then searches and aligns the sequence along a specified genome [1]. This is often used for finding a common evolutionary relationship between two sequences, but can also be used to retrieve the original genomic location of a sequence along a genome.

Part II

The project

Chapter 3

Method

In this chapter we will first introduce the problems we are trying to solve, and describe some requirement for our system. Then we will look into some of the technological approaches we will use to implement this system. Finally, we will describe a few test sets which will be used to test the application.

3.1 Problem description and requirements

3.1.1 Limitations of benchmarking in bioinformatics

Even though there exists good benchmarks for certain problems in bioinformatics, there is still room for improvements in many areas. Here we will discuss some of the current issues which we will try to address in this thesis.

3.1.1.1 Availability

The first thing to notice about benchmarking in bioinformatics is that it seems to be almost non-existent in certain areas. For instance in cases like read mapping there have been done little research into the benchmarking and the evaluation of different algorithms [20]. For other problems like gene prediction, the benchmarks that do exists are so outdated or irrelevant for a users need, that they are of little use.

Though it really depends on the problem area. For instance in other cases like multiple sequence alignment there are several benchmarks already available [2].

3.1.1.2 Development time

One reason that benchmarks seems to be lacking in certain areas, might be that benchmarks can be quite difficult and time consuming to develop. If someone want to create a new benchmark they first have to create the test sets which are of good quality and relevance for the algorithms they wish to test.

Additionally they need to do the coding, evaluate the results in a proper way, and make it available to the public. In the end, many researchers will find that the development time of a benchmark will not be worth the effort, and therefore might refrain from the idea of developing new benchmarks.

3.1.1.3 Quality of test sets

Another problem is that the test sets created are not always as good or relevant as they should. When looking at a test set there are several things to consider:

- Are the test sets relevant to what we are looking for in a program?
- Do the test sets actually contain quality data that tests the algorithms in a way that is both thorough and realistic?
- Are the test sets completely fair, or do they have some bias which is especially favorable for a certain kind of algorithm?

The creation of test sets will be outside the scope of this project however, and we will leave this part up to the users of the system.

3.1.1.4 Quality of evaluation

When it comes to evaluation of the results we must also consider:

- Are the results easy to understand and are they properly explained?
- Do they have the right emphasis on what's important?
- Are the results relevant to what you are looking for in a program?
- Are there important measurements that are missing?

3.1.1.5 Ease of use

Another problem is all the work that have to be done for users who want to evaluate a set of algorithms. First they have to find a benchmark that is relevant for their work, which may or may not exist. Then they have to download, set up and run the benchmark on the algorithms they wish to evaluate, which often is undocumented and hard to understand.

Next they have to evaluate the results given by the algorithms, which all comes in different formats that has to be understood and evaluated in a proper way. Finally they need to make sense out of the benchmark results, which can be difficult to understand if not explained properly.

3.1.2 Introducing dynamic benchmarking

To address some of the issues discussed above, we will be introducing a new dynamic benchmarking system.

The dynamic benchmarking system will be developed as an online web application, which streamlines each step of the benchmark process. This

system will provide benchmark creators and benchmark users a common platform for creating, distributing and evaluating test sets. The goal is that the system should provide the following core functionality:

- **Sharing with community:** Benchmark creators should be able to upload and share their test sets with the community.
- **Integrated evaluation:** It will contain integrated evaluation functionality for each supported problem area, so there will no longer be a need for creating the same evaluation functionality from scratch.
- **Integrated algorithms:** One goal is to provide a set of popular algorithms for each supported problem area, so the users won't have to set it up themselves.
- **Flexible:** The system should support several types of evaluations, problem areas, test sets, file formats and benchmark types using the same streamlined approach.
- **Extendable:** The system will be free and open source, encouraging extensions of new functionality from the community.

3.1.3 Requirements

Not all of the functionality mentioned in the previous section will be implemented to its full extent, but the goal of this system is to create a foundation for most of them. Here we will look at some specific requirements that should be implemented in this system.

3.1.3.1 Supported problem areas

The benchmark system will support all the problem areas introduced in section 2.3; gene prediction, motif discovery, nucleosome prediction and splice site prediction. Since these problem areas share the same input and output, they share pretty much the same functionality when it comes to creating and evaluating benchmarks.

3.1.3.2 Supported benchmark test sets

To give the benchmark creator some flexibility in how they wish to define their test set, we will offer a selection of different types of test sets:

- **Sequence level:** Here we have a test set with a set of sequences, some that contains a pattern and some that don't. With sequence level test set we want to check the algorithms capability of finding patterns in the right sequences.
- **Nucleotide level:** This test set also contain a set of sequences, but here we are interested in finding out if the algorithms are able to find the exact location of the pattern along the sequence.

- **Base pair probability level:** These test sets are meant for the base pair probability prediction algorithms we introduced in section 2.4.2, and are therefore somewhat different from the others. Here we have a test set of segments and we want to predict the probability to find a pattern for each base pair in these segments.

There are some advantages and disadvantages one must consider when choosing either nucleotide level and sequence level benchmark. The advantage of using nucleotide level benchmark is that it is very precise. It will tell you exactly how good an algorithm is to find the exact position of a pattern.

On the other hand when dealing with nucleotide level benchmarks we often run into the troubles with imbalanced data, with an overwhelming amount of negatives. Sequence level benchmark has the advantage of diminish the amount of negatives in a test set, bringing the negative/positive ratio to a normal level.

In addition to the types of test sets we introduced above, we also will distinguish between two different types of benchmarks:

- **Single test set:** Contains just a single test set file, which is a good option for simple benchmarks for fast creation and evaluation.

- **Benchmark suite:** Should contain two or more files. This is aimed at large benchmarks that want a thorough evaluation of all the algorithms strong and weak points. The number of test sets varies, but large benchmark suites usually contain several hundred test sets.

3.1.3.3 Supported file formats

We won't have any integrated algorithms in our implementation because there will simply not be enough time to implement it. We do however support the output files from a selection of popular algorithms, to ease the work of the benchmark user to some degree.

For gene prediction we will support the result from Glimmer, Prodigal and genemark. For motif discovery we offer support for YMF, Weeder and MEME. All these algorithms was introduced in section 2.3.

3.2 Technological approach

In the last section we explained the dynamic benchmarking system we wish to implement. Creating such a system from the ground however would be too ambitious for this project. Luckily there already exists a lot of open software and tools out there which we can build upon and use to create this system. In section 2.5 we introduced some of the relevant technologies, in this section we will explain how we can use these technologies to implement our system.

3.2.1 The framework

The system will be implemented using the Galaxy framework. Galaxy will provide an excellent framework for our system for several reasons:

- **Aimed at biologists:** First of all the, the tools that Galaxy provides can be used by anyone without any programming experience, making it ideal for biologists [16].
- **Integrated sharing:** In addition, Galaxy is a platform which encourages an open community, as every result returned by a Galaxy tool can easily be shared to others. This makes it easy to reproduce the results of others, and offers an excellent feature for sharing benchmarks with the community [16].
- **Open source:** And finally, as Galaxy is an open platform, its possible to extend its features and add new functionality [16].

The system won't be implemented as part of Galaxy however, because it miss some useful components that Hyperbrowser provides for us. One of the main goals of Hyperbrowser was to facilitate sophisticated statistical analysis of genomic tracks, which is lacking in Galaxy [32]. This statistical functionality will come in handy when implementing evaluation functionality, where we want to compare prediction with answer tracks.

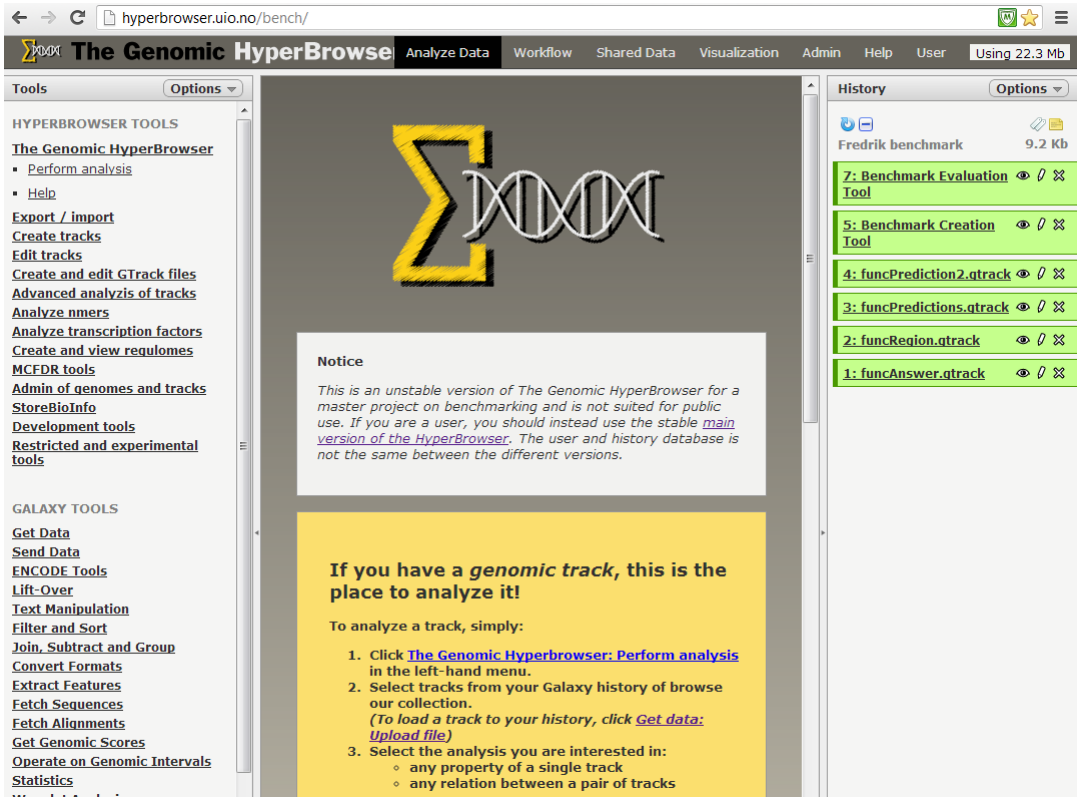

Figure 3.1: The Genomic Hyperbrowser web application.

As the system will be implemented as part of Hyperbrowser which is based on the Galaxy framework, we will now explain some of the terminology used within the system:

- **Tools:** Pretty much all functionality within Hyperbrowser is defined by a tool. All the tools available can be found at the left pane of the web application. Here we can see the Galaxy Tools at bottom (the tools provided by Galaxy), and the Hyperbrowser Tools at the top (the tools provided by Hyperbrowser). These tools have different purposes, from doing some analysis or statistic on different genomic track, to the uploading or downloading of different data. What they all got in common is that tool ask for some input, and once the input is specified and you hit execute, it will start a job running on the server.

- **History element:** Once the job is complete, it will be saved as a History Element which are displayed as either green (on success) or red (on failure) links on the right pane. Here we can download or watch the results, and look at debug information etc.

- **History:** All the history elements will be added to the history, as some tools depend on the results from previous tools (ie. benchmark evaluation depends on a benchmark specification, which in turn depends on test sets). The history can be shared to the community, which makes distributing benchmarks and reproducing results an easy task.

- **Hyperbrowser statistics:** Hyperbrowser comes with sophisticated statistical analysis of genomic tracks, and has already 62 descriptive statistics as part of its system [32]. As these statistics are meant for statistical analysis and large amounts of data, they use Numpy arrays, which was introduced in section 2.5.1.1, to speed up the calculations. This statistical functionality is useful in the evaluation process, when comparing the results with an answer.

3.2.2 Representing test sets

For the test set, and result files we use the GTrack format which was introduced in section 2.5.4. The reason for choosing GTrack is first of all because it's the main format used in Hyperbrowser already. As well as it's a format that supports 15 different track types, making it a format that can represent a good selection of different of data [17]. This is a useful feature as it will support different problem areas which might be added later.

Figure 3.2 shows an example of a simple GTrack file. The header first defines one of the 15 track types supported, in this case "valued segments". Then it defines the "value type" of the value column, which in this case is "binary". Next it consists of a column specification line, which specifies all the columns in the file; seqid, start, end, value and strand. And finally a set of data lines, containing the actual data as specified by the specification line.

```
# GTrack
##track type: valued segments
##value type: binary
###seqid          start    end      value    strand
chr1     0        20       1        +
chr2     50       70       0        -
```

Figure 3.2: An example of a GTrack file.

The strand column contains either a plus or a minus sign, and tells us the direction of the sequence. The plus sign indicates the forward strand, while the minus sign indicates the reverse strand. See figure 2.1 in section 2.1.1 for more information.

There is an important difference with our approach compared to the way that test sets have been defined in the past. Normally, the test sets are defined by the sequences in either FASTA or similar formats. In our case however, we define our test set with the segments of the sequences.

There are some advantages with this approach. First of all, retrieving the sequences based on the segments in FASTA format is already supported in Hyperbrowser. Going the other way from sequences to segments is more complicated, as it would require some sequence alignment tool such as BLAST to retrieve the segments. Secondly, it's a format that offers more support for different data types, and is more flexible in what data we can represent which is vital for our system. And finally, it's a format that takes less space, is more compressed and contain useful information which are often excluded in the FASTA format.

3.2.3 Representing sequence data

For representing the sequences that will be given as input for the algorithms we use the FASTA format which was introduced in section 2.5.5.

```
>chr1:0-20
CGGTCAGCTTCGATGTTACT
>chr2:50-70
AATCGGTTACGGATTACGTA
```

Figure 3.3: A FASTA file containing the sequences defined by the GTrack file in figure 3.2.

When we retrieve a test set from Hyperbrowser it will look similar to the example in figure 3.3. This FASTA file contain two sequences. Each sequence is separated by a description line, starting with a ">" sign to define each sequence. The description line should be unique, and can contain any information we like. When we retrieve the FASTA file from hyperbrowser, the header contain the chromosome, and the start and end positions of the sequence, as shown in figure 3.3.

As the description line of each sequence, along with the start and end positions for each predicted segment is often included in the result files generated by the algorithms, we can convert the result files back to the GTrack format for evaluation. We will look more into the conversion from result files to GTrack files in the next section.

3.2.4 Result file conversion

As every result file from different algorithms are different, we need some way of converting these formats back to the original GTrack format for evaluation. To achieve this we use Python regular expressions, introduced in section 2.5.1.2, to extract the desired information needed to convert the file to GTrack.

As an example, figure 3.4 shows a few lines from the MEME motif discovery algorithms result file, which contains the relevant data needed to convert it back to GTrack.

```
Sequence name  Start  P-value        Site
-------------  -----  -------  ----------------------
chr1:0-20          8  8.48e-27  CGGTCAGC TTCGATGT TACT
chr2:50-70         4  2.45e-26   AATC GGTTACGG ATTACGTA
```

Figure 3.4: Part of a result of file of the MEME motif discovery algorithm.

To retrieve the relevant information from the text file we can create a regular expression: (.*)\s+(\d+)\s+(\d*\.\d+e-\d+)\s+[ACGT]+\s+([ACGT]+). Regular expressions are known to be rather cryptic, so table 3.1 gives a description of each pattern included in this regular expression.

Pattern	Description
.*	Any sequence of text, except new lines (ie. "chr1:0-20")
(.*)	With parenthesis around the expression, we can retrieve the data later.
\s+	One or more space or tabs.
\d+	One or more digits.
\d*\.\d+e-\d+	Pattern to match the p-value, ie. 8.48e-27.
[ACGT]+	One or more of the characters ACGT (ie "TTCGATGT").

Table 3.1: A few regular expression patterns.

Parsing the result file in figure 3.4 with the regular expression, we can first retrieve the chromosome and start and end position with the (.*) pattern to retrieve the text "chr1:0-20". Secondly, we can retrieve the start segment of with the pattern (\d+), where we retrieve the digit "8". Next, we can extract the p-value with the pattern (\d*\.\d+e-\d+), and retrieve the value "8.48e-27".

Finally, we can find the length of the pattern by retrieving the sequence "TTCGATGT" with the ([ACGT]+) pattern, and measure its length, which

in this case is eight. With this information we can convert the result file back to GTrack as we can see in figure 3.5.

```
# GTrack
##track type: valued segments
###seqid          start    end      value     strand
chr1     8        16       8.48e-27          +
chr2     54       62       2.45e-26          +
```

Figure 3.5: The MEME result file from figure 3.4 converted to GTrack.

3.3 Benchmark test sets

For testing the implementation we wanted to find relevant test sets for different cases of benchmarking. They are not meant to be realistic benchmark cases where we optimize parameters and compare several algorithms, but rather to serve as a proof of concept for the benchmarking implementation.

We've found test sets for both sequence level, nucleotide level and base pair probability level. Additionally we've found test set across different problem areas, both for motif discovery and gene prediction. And finally we've made both single version test sets and benchmarks suite test sets. This should provide a thorough test of the entire application, testing all its functionality.

As all existing test sets exists in either FASTA or similar formats, we needed to find a way of converting these files to our GTrack format. To achieve this we can use the Basic Local Alignment Search Tool or BLAST which was introduced in section 2.5.6. By feeding this tool the sequences from the test sets we can, given a perfect match, retrieve the chromosome, start and end values which are needed to convert the results to the GTrack format to define our test sets.

3.3.1 Motif discovery benchmark on sequence level

For the benchmark test sets on sequence level we used the test sets from the article "Sequence and chromatin determinants of cell-type–specific transcription factor binding" [4]. These test sets consists of 596 files, half of which are used to train a Support Vector Machine algorithm and the other half used for testing the algorithms. Each of these test sets contains around 100 to 2000 sequences of length 100.

The test set only contain the sequences however, with no genomic location which are needed to convert the test set to GTrack. Luckily, with a quick exchange of emails with one of the authors we received to locations and could create a test set for our system.

Based on these test sets we created two sets of test sets to test our application. One single test set consisting of 200 sequences, and one

benchmark suite with twenty individual test sets ranging between 100 and 365 sequences each.

3.3.2 Motif discovery benchmark on nucleotide level

For the benchmark test sets on nucleotide level we used the test sets from the article "Assessing computational tools for the discovery of transcription factor binding sites" [40]. There are 52 FASTA files containing sequences from various of different species.

To find the genomic location of these FASTA sequences we used BLAST. The results from BLAST was then converted to GTrack to create our test set.

We tested the application with one single test set, and a benchmark suite consisting of 26 test sets.

3.3.3 Gene prediction benchmark on nucleotide level

Finding a test set for Gene Prediction that also included the actual answer segments posed to be a bit of a challenge. Eventually though, we ran across the article "Evaluation of Gene Structure Prediction Programs" from 1996 [7], which also provided the answer segments.

We ran the FASTA sequences through BLAST but found that many of the sequences were broken as the test set is quite outdated. We did however collect 103 sequences from the human genome which was still intact, and converted these to GTrack to make a test set.

3.3.4 Base pair probability level benchmark

For base pair probability level, we used the Vitamin D Receptor (VDR) binding sites for the entire chromosome 21 as a test set, which consists of almost 47 million base pairs. The reason for choosing such a big test set is that the number of positives is so few and far between (only 35 in this test set), that we need a test set of this size in order to get any useful results.

Chapter 4

Implementation

In this chapter we will look at the implementation of the benchmarking system. First we'll describe the system workflow from a users point of view, where we describe each step of the benchmarking process. Secondly we'll look into the system architecture, where we describe how the system is built.

4.1 Program workflow

We choose to explain the workflow in two separate parts, as the system is aimed at two different users, the benchmark creator and the benchmark user.

4.1.1 Benchmark creator workflow

The benchmark creator has the job of creating test sets, upload them to Hyperbrowser, create a benchmark specification and make it available to other users. This process outlined in figure 4.1, and in this section we will explain each of these steps.

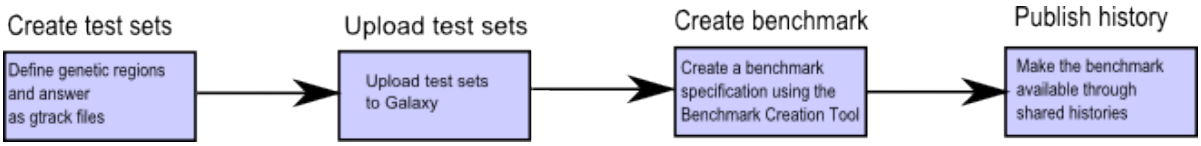

Figure 4.1: The benchmark creator workflow

4.1.1.1 Creation of test sets

A test set can be created using different approaches depending on the problem area, and it's really up to the benchmark creator which ones he prefers. One common approach is to extract regions and answers from a database, for instance TRANSFAC for motif discovery [42]. However the creation of test sets goes beyond the scope of this thesis.

When we define test sets its important that its in a compatible format. There are currently two formats which can be used for creating test sets:

- **GTrack:** This is the main format we use which was introduced in section 2.5.4.
- **BlastHit:** An alternative format is the BLAST hit tables generated by the Basic Local Alignment Search Tool. This way we can use the output from BLAST directly, without having to convert them to GTrack first.

Furthermore, there are differences depending on what kind of test sets you wish to define:

- **Nucleotide and base pair probability level:** For these kinds of test sets we need two separate files, one to specify the regions of the test sets, and a second one to define the answer. We can see examples of a test set and an answer file in the figures 4.2 and 4.3 respectively.

```
# GTrack
##track type: segments
###seqid        start    end      strand
chr1    0       20       +
chr2    50      70       +
```

Figure 4.2: A test set defining two DNA segments.

```
# GTrack
##track type: segments
###seqid        start    end      strand
chr1    1       3        +
chr1    14      17       +
chr2    56      60       +
```

Figure 4.3: An answer file which defines the answer segments within the segments specified in the test set in figure 4.4.

- **Sequence level:** Test sets on sequence level should just be one file GTrack file containing the sequences, and a binary value telling us whether or not a pattern exists within the sequence or not. Figure 4.4 shows an example of a test set for sequence level benchmarks.

4.1.1.2 Uploading test sets

Once the test sets are created, they need to be made available to Hyperbrowser. This can be done in one of two ways depending on if its a single test set benchmark or a benchmark suite:

```
# GTrack
##track type: valued segments
##value type: binary
###seqid          start    end      value    strand
chr1     0        20       1        +
chr2     50       70       0        +
```

Figure 4.4: A test set on sequence level, specifying the DNA segments if there exists patterns inside it.

- **Single test sets:** For uploading a single test set, we use the "Get data -> Upload file" tool within Hyperbrowser. Here the benchmark creator specify the file to be uploaded, its file format, the genome and hit execute to upload the file. This file will now be made available as a history element.
- **Benchmark suites:** For uploading benchmark suites, all the test sets must first be compressed into a tar archive. For nucleotide level benchmarks we need to create two archives, one for the regions and one for the answers. As Hyperbrowser do not allow us to use the "Upload file" tool for zip files, in order to prevent people from using it as an online storage area, we needed to find another way of uploading it to Hyperbrowser. The solution is to upload the tar archives to a server where it can be retrieved using an URL, which can be used to download the test sets later on.

4.1.1.3 Creating a benchmark specification

Once the test sets are uploaded, the benchmark creator need to create a benchmark specification. This is achieved using the "Benchmark creation tool" under "Restricted tools" in Hyperbrowser. Here the benchmark creator creates a benchmark specification by filling in the following:

- **Genome build:** Specifies the genome of the test set.
- **Select problem area:** Specifies the problem area the benchmark creator wants to benchmark, which can either be motif discovery, gene prediction, splice site prediction or nucleosome prediction.
- **Test set type:** Specifies the test set type, which can either be nucleotide level, sequence level or base pair probability level.
- **Benchmark type:** Specifies the type of benchmark the benchmark creator wants to create, which can either be Single test set or Benchmark suite.
- **Select test set:** Selects the test set file(s) which specifies the genomic regions of the test set. With single test sets we can select the file from history, with benchmark suites the benchmark creator need to specify an URL pointing to an uploaded tar archive.

- **Select answer:** Selects the answer file(s) which contains the answer to our test set. This is only applicable to Nucleotide Level and Base Pair Probability Level, as Sequence Level has the answer included in the test set.
- **Select feature track:** This option is only available for Base Pair Probability Level and specifies the input genomic track for base pair probability level prediction algorithms.

Once the entire form is filled, the benchmark creator can hit execute. The benchmark specification will then be stored in the Hyperbrowser history and is ready for use.

4.1.1.4 Publishing a benchmark

Distributing the benchmark can be done easily in Hyperbrowser using the publish history function. This way, the history will be made available for other Hyperbrowser users to import. Publishing the history is achieved by following these steps:

1. **Change history name:** First the benchmark creator should choose a descriptive name for their benchmark, (default is "Unnamed history"). The name should at least specify problem area, benchmark type and the genome.
2. **Publish:** Click the "Options" button, and click "Share or publish". On the next page click "Make History Accessible and Publish". Now the history can be imported by other users by clicking "Shared data" and "Published histories" at the top pane.

4.1.2 Benchmark user workflow

The benchmark user has the task of setting up and running test sets on a set of algorithms and have them evaluated. Figure 4.5 displays how this process is carried out in our system, and in this section we will explain each step of the benchmark user.

4.1.2.1 Importing a benchmark

Assuming the benchmark user haven't created a benchmark specification by himself, he can get one by importing a benchmark specification to his history. This is done by clicking on "Shared data" and "Published histories" in the top pane in Hyperbrowser. Select one of the available benchmark specifications, and click "Import history" in the top right corner, and the benchmark specification is then imported to history.

4.1.2.2 Retrieving a test set

Once the benchmark specification is imported to history, the benchmark user can download the test sets using the "Benchmark Retrieval Tool"

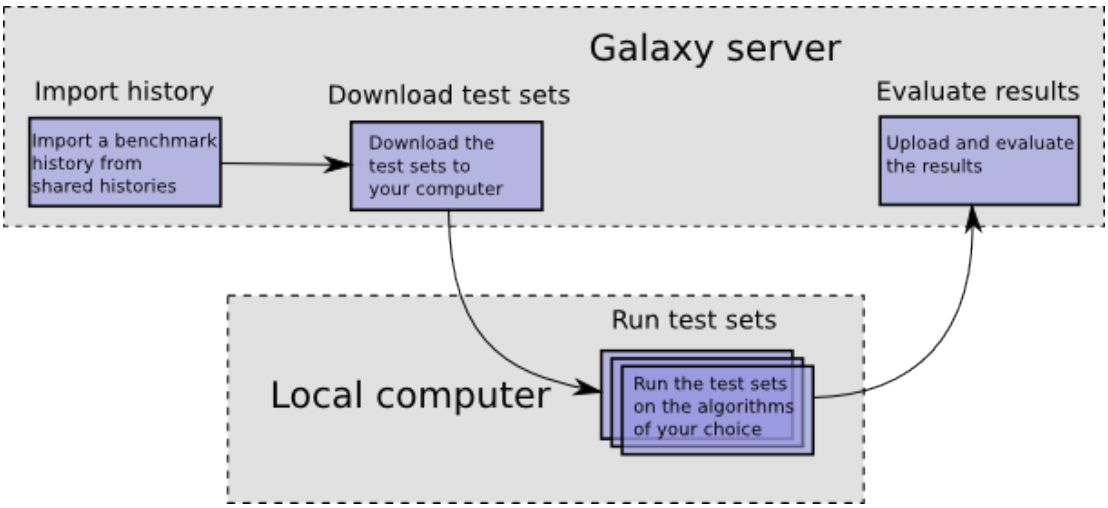

Figure 4.5: The benchmark user workflow

under "Restricted Tools". Here we just select the benchmark specification created by the "Benchmark Creation Tool" and hit execute. Once the job is finished, the benchmark user can download the test sets by clicking on the finished job and click the "save" icon.

4.1.2.3 Running test set on algorithms

The next step is to run the test sets on different algorithms and collect some result data. Running the test sets on a selection of algorithm can be a challenging and time consuming process, especially if documentation is poor and the results are difficult to understand.

Once we have run the test set on a selection of algorithms and retrieved the results, it's important that the result files are in a compatible format before we upload. We do support a few outputs from a selection of algorithms as we presented in section 3.1.3.3. If the format is not supported, they need to be converted to GTrack manually.

For sequence and nucleotide level benchmarks, the result files should look something like figure 4.6. Here the start and end defines the segments which are predicted by the algorithm. The value column contains the score value that the algorithm have given the segments.

```
# GTrack
##track type: valued segments
###seqid        start    end      value    strand
chr1    2       5        0.9      +
chr1    15      18       0.85     +
chr2    56      60       0.8      +
chr2    64      69       0.95     +
```

Figure 4.6: A result file for nucleotide or sequence level in GTrack format.

```
# GTrack
##track type: function
###value
####seqid=chr1; start=0; end=20
0.52212
0.83927
.
.
.
0.13732
####seqid=chr2; start=50; end=70
0.32212
0.53927
.
.
.
0.73732
```

Figure 4.7: A result file for base pair probability level in GTrack format.

For base pair probability level the result are given as a function as displayed in figure 4.7, where each base pair in the test set receives a score.

4.1.2.4 Uploading result files

Once the result files are ready the next step is to upload them to Hyperbrowser. This step is identical to the uploading of data sets explained in section 4.1.1.2, by using the "Upload File" tool for single test sets. And for benchmark suites we make a tar archive, uploading it to a server where we can retrieve it via a URL.

In the case of a benchmark suite it's important that the tar archive follows certain conventions, so there won't be any blunders in the evaluation process:

- First of all, the result files for each algorithm should lie in a separate sub directory so the results are not mixed with each other.
- Secondly, its important that the results files are named in the same order as the test sets, otherwise its impossible to know what result belongs to what test set.
- And finally, make sure there are no result files missing. Each sub directory should have the same number of result files as the number of test sets defined in the benchmark suite.

4.1.2.5 Evaluating results

Finally, the benchmark user can evaluate his result using the "Benchmark Evaluation Tool" under "Restricted and experimental tools". In this tool

one must first select the benchmark specification, and result location (from history or from URL). Next, select the result files, either from the history or from the specified URL and hit execute. Once the job is finished we can view the results by clicking on "the eye" icon on the resulting history element.

4.2 Architecture

In this section we will look into the architecture of the benchmarking system, which we can see in figure 4.8. This only includes the features implemented for this benchmarking system, and not the functionality already provided by Hyperbrowser.

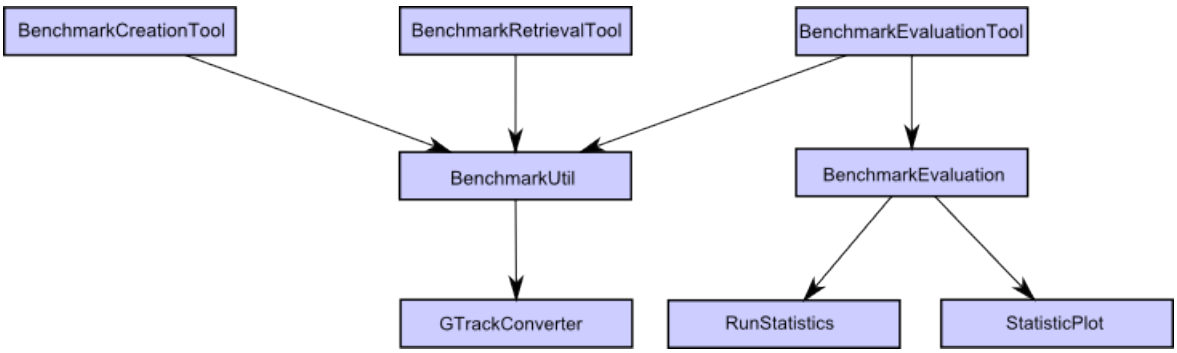

Figure 4.8: Overview of the benchmark evaluation architecture

The most important thing to consider when designing this system is that it should be easy to add more components later on, whether it be new file formats, utility functions, or evaluations. There have been done a lot of changes to the architecture underway, as more and more features was added, and it was quite a challenge to make a good and consistent architecture.

4.2.1 Benchmark tools

The benchmark tools serves as the top layer, and has the job of handling the input and output of the user. There are three benchmark tools implemented in this project, which handles three essential parts of the benchmark process. Each of the three tools are described below:

1. **Benchmark Creation Tool:** Creates a benchmark specification based on input from the user, which includes the test sets, genome, problem area etc. This benchmark specification is then saved, and will be stored in the history element to be used later from the tools below.

2. **Benchmark Retrieval Tool:** Takes a benchmark specification from BenchmarkCreationTool and retrieves the test set defined in the specification. This can either be a single test set or a zip file containing a benchmark suite.

3. **Benchmark Evaluation Tool:** Takes a benchmark specification and one or more result files as input. Then it runs evaluations on the predictions as specified in the benchmarked specification and prints out the benchmark results.

4.2.2 Benchmark util

The tools we introduced above shared some of the same functionality, so to avoid duplication of code a utility class was created to serve as a common service class for all the tools to use. This shared functionality typically involved one of these tasks:

- **File format conversion:** Handles files or set of files which are in an incompatible format, and passes it to GTrackConverter for conversion to the GTrack format. This is useful both for benchmark creation and benchmark evaluation.
- **Downloading file from URL:** Hyperbrowser does not allow uploading of zip files in their upload file tool to prevent people from using it as a storage area. So in the case of benchmark suites which involves several test sets, we instead let people specify an URL pointing to their zip file, which will then be downloaded using the "wget" command. This functionality is useful both for benchmark creation for uploading test sets, and benchmark evaluation for uploading results.
- **Extracting tracks:** This involves handling the extraction of files (usually FASTA sequences) from within the Hyperbrowser system, as specified by the segments of a GTrack file. This is first and foremost useful for benchmark retrieval, but can also be useful for converting certain file types to GTrack.

4.2.3 GTrack converter

One of the goal of the benchmarking system is to support a wide variety of different algorithm formats to ease the work for the benchmark user. GTrackConverter was implemented just for this purpose, and has the job of converting different file formats to the GTrack format. This is achieved by using python regular expressions introduced in section 2.5.1.2 to extract relevant data from a given file.

Currently GTrackConverter supports the output from the following algorithms for motif discovery and gene prediction:

- **Motif discovery algorithms:** MEME, Weeder and YMF.
- **Gene prediction algorithms:** Glimmer, Prodigal and Genemark

Additionally GTrackConverter also support the BLAST hit tables as mentioned earlier in section 4.1.1.1. This allows the benchmark creator to create a benchmark test set using the output from the Basic Local Alignment Search Tool.

4.2.4 Benchmark evaluation

BenchmarkEvaluation has the task of running statistics on the predictions given as input, where it is compared with an answer given by the benchmark specification. These results are then given to StatisticPlot for the creation of different result plots.

Currently, there are three different evaluations implemented in this class :

1. **Binary classification benchmark:** This evaluation is meant for binary classification problems benchmarks with single a test set. It runs two statistics:

   (a) **Overlap statistic:** The first statistic compares the overlap between the prediction and the answer, these statistics will be introduced in section 4.2.5.1 and 4.2.5.2.

   (b) **ROC statistic:** The second statistic calculates data required to create a ROC curve, these statistics will be introduced in section 4.2.5.3 and 4.2.5.4.

2. **Binary classification benchmark suite:** This evaluation runs the same statistics as the above evaluation, but on several test sets. Here we collect results from every single test set, as well as global results based on the results from all test sets.

3. **Base pair probability benchmark:** This evaluation is meant for the base pair probability predictions, where we evaluate a function instead of segments. This evaluation runs three statistics:

   (a) **Difference comparison statistic:** The first statistic computes the average scores outside and inside answer segments, and will be introduced in section 4.2.5.6.

   (b) **ROC statistic:** The second statistic calculates the data required to create a ROC curve, and will be introduced in section4.2.5.5.

   (c) **Score distribution statistic:** The third statistic calculates the score distribution of the function and the answer, and will be introduced in section 4.2.5.7.

4.2.5 Statistics

The actual evaluation, where we compare result tracks with answer tracks, is implemented using Hyperbrowsers statistics which we introduced in section 3.2.1. These are essentially python classes that do computation on genomic tracks, and should follow certain conventions. Statistics are not called from BenchmarkEvaluation directly however. They are called through a function runManual in the class GalaxyInterface, which will preprocess and store the data from the given tracks in numpy arrays.

The benchmarking systems uses seven different statistics (six of which was implemented during this project) to evaluate benchmarks which we will introduce here.

4.2.5.1 RawOverlapStat

This statistic computes the overlap of two segment tracks (the answer and the prediction) on nucleotide level, meaning it compares every base pair in the segments and categorize it as either:

- **True Positive (TP):** If the base pair is within a segment in both the answer track and the prediction track.
- **True Negative (TN):** If the base pair is not within a segment in both the answer track and the prediction track.
- **False Negative (FN):** If the base pair is within a segment specified by the answer track but not in the prediction track.
- **False Positive (FP):** If the base pair is within a segment specified by the prediction track but not in the answer track.

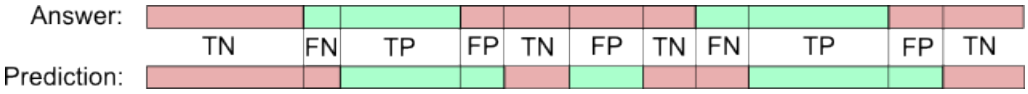

Figure 4.9: A prediction track is compared with an answer track.

The statistic counts the number of occurrences of each class, and returns the number of each TN, TP, FN and FP. This statistic was already implemented prior to this project.

4.2.5.2 SequenceLevelOverlapStat

This statistics has the same job as RawOverlapStat, except it computes the occurrences of TN, TP, FN and FP on sequence level. Meaning that the test set only specify a binary answer which indicates whether the sequence contains a pattern or not, but it does specify the exact segment within the sequence.

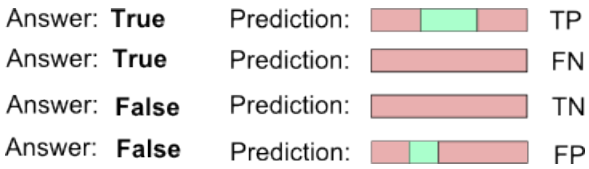

Figure 4.10: Prediction segments are categorized based on a binary answer.

So if the prediction specify one or more segments within a binary "True" answer we increment TP by one, or if "False" we increment FP by one etc. When all sequences are categorized it returns the number of occurrences for each class, just like RawOverlapStat.

4.2.5.3 MarksSortedNucleotideLevelSegmentsStat

This statistic creates data which will later be used to create ROC curves as introduced in section 2.4.1.2, on nucleotide level. Here it goes through each base pair and stores data in a numpy array. These data are as follows:

- **Score:** For an algorithm to be compatible with ROC curves it need to score its segments with a score value. Meaning every segments receives a score, and the higher the score the higher the algorithm rates the probability of this segment to contain a pattern. Base pairs within these segments receive the same score value as the segment, whereas base pairs outside these segments receives a score of zero.
- **Random number:** As the numpy array will be sorted later on based on the ranking value, we stumble onto the problem that many of the ranking values are the same. To solve this problem we add this random number to add some random chance to the sorting process.
- **Binary value:** A binary value which is 1 if the base pair is within an answer segment and 0 otherwise.

Score	Random	Binary		Score (sorted)	Random (sorted)	Binary
0	0.2	0		0.9	0.9	1
0.8	0.9	0		0.9	0.7	1
0.8	0.1	0		0.9	0.3	0
0.8	0.7	1		0.9	0.2	1
0.8	0.4	1		0.8	0.9	0
0.8	0.8	1		0.8	0.8	1
0	0.3	0		0.8	0.7	1
0	0.9	1		0.8	0.4	1
0.9	0.7	1		0.8	0.1	0
0.9	0.2	1		0.7	0.8	1
0.9	0.9	1		0.7	0.7	0
0.9	0.3	0		0.7	0.2	1
0	0.1	0		0	0.9	1
0	0.8	0		0	0.8	0
0	0.0	0		0	0.6	0
0.7	0.7	0		0	0.5	1
0.7	0.8	1		0	0.3	0
0.7	0.2	1		0	0.2	0
0	0.5	1		0	0.1	0
0	0.6	0		0	0.0	0

Table 4.1: Shows example data for ROC curves before and after sorting.

Table 4.1 outlines an example of what this numpy array might look like, though in reality these arrays are usually thousands or millions elements long. The next step is to sort this array, first based on rank number and secondly the random number, as we can see in the columns to the right.

This way the base pairs which the algorithm gives the highest rating are located at the start of the array, while the lowest are at the end. A sorted array with the binary 1's gathered at start and the binary 0's at the end would therefore indicate a good prediction, whereas random placing of 1's and 0's would indicate a random prediction. We will see more of this while we create ROC curves in section 4.2.6.2.

4.2.5.4 MarksSortedSequenceLevelSegmentsStat

This statistic has the same job as the above statistic, except this statistic computes the data on sequence level. The data will look the same as displayed in table 4.1, except the data represents something different:

- **Score:** The highest score value given by the algorithm in this sequence. If prediction is found within the sequence, the score is set to 0.
- **Random value:** Serves same purpose as nucleotide level.
- **Binary:** Tells whether or not the sequence contained a pattern. 1 means a sequence with a pattern, and 0 a sequence without a pattern.

This statistic therefore returns identical output as the nucleotide level version, and the ROC curves can be generated in the same way.

4.2.5.5 MarksSortedProbabilityLevelSegmentsStat

This statistic computes the values for a ROC curve on base pair probability level. It's quite similar to the nucleotide level version, but it doesn't have to consider the start and end of prediction segments as the answer is one continuous function. The result array has the same values as the above two tools.

4.2.5.6 FunctionTrackAnalysisStat

This simple statistic collect data that will be used for evaluation of benchmarks on base pair probability level as introduced in section 2.4.2.1. It calculates and returns four different values:

- The number of base pairs outside and an answer segment (22 in figure 4.11).
- The number of base pairs within an answer segment (8 in figure 4.11).
- The sum of all values outside an answer segment.
- The sum of all values inside an answer segment.

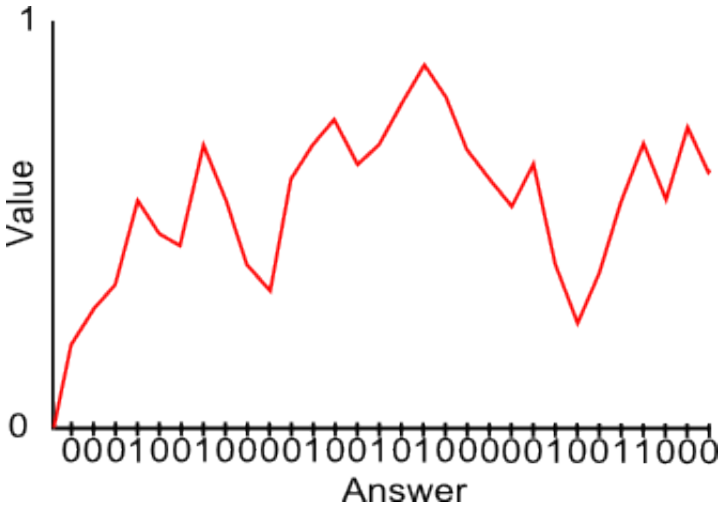

Figure 4.11: Displays a function, with the answer shown below.

4.2.5.7 FunctionTrackScoreDistributionStat

This statistic computes the score distribution of both the entire function and the answer segments, as introduced in section 2.4.2.2. This is achieved by creating one array for each function of length 101, whose values are incremented by one for each score (with scores ranging between 0.00 to 1.00).

To solve the issue of imbalanced data between the two arrays, the number of total function values and answer values are returned as well. These will later be converted into percentages so that the values of the two functions can be compared.

4.2.6 Evaluation plots

When we have the results from all the statistics, we can proceed and turn this data into something we can present to the user. This class serves the purpose of taking the results from the statistics, calculate different measurements and create plots created by the matplotlib library introduced in section 2.5.1.3. The benchmarking system currently have four kinds of statistic plots which will be described next.

4.2.6.1 Binary classification statistics

This statistical plot takes a set of TP, TN, FP and FN from either RawOverlapStat or SequenceLevelOverlapStat and calculates a set of different measurements. These measurements are the same measurements which was introduced and explained in section 2.4.1.1.

The calculations of these measurements is pretty straight forward, and was outlined in table 2.2 in section 2.4.1.1. The result will be saved as png file like the one displayed in figure 4.12, as well as the actual values in html table.

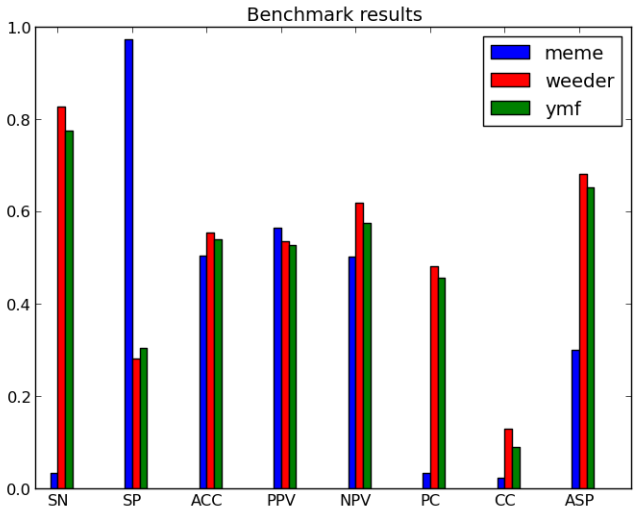

Figure 4.12: A statistic plot showing the result of a binary classification benchmark

4.2.6.2 ROC curve

This statistical plot takes a set of numpy arrays returned from either MarksSortedSequenceLevelSegmentStat, MarksSortedNucleotideLevelSegmentStat or MarksSortedProbabilityLevelSegmentsStat, and creates a ROC curve.

First of all the array is first sorted by rank, as displayed in table 4.1, and the binary values are plotted as displayed in figure 4.13. A binary 1 moves the curve up along the Y axis, while a 0 moves the curve along the X axis.

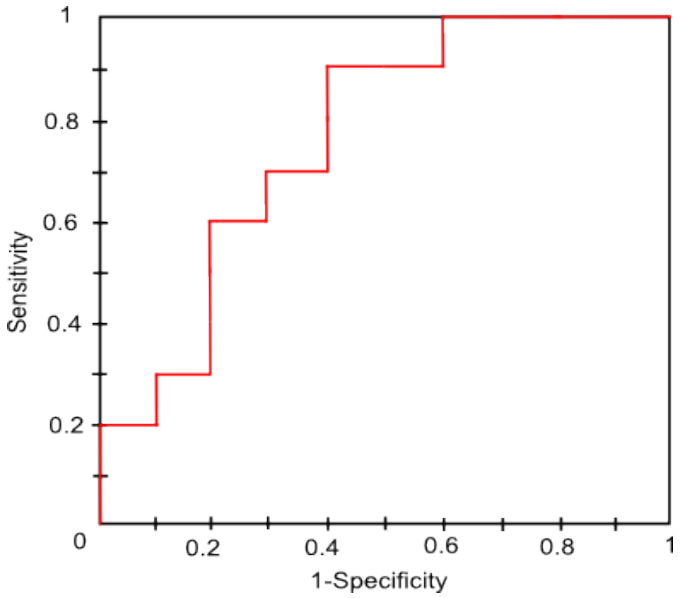

Figure 4.13: A ROC curve created based on the values from table 4.1.

A perfect prediction will therefore follow rank all the answers highest, and will follow the Y axis all the way up before it will start to follow the X axis. The prediction shown in figure 4.13 which is based on the values in table 4.1 are quite okay, not perfect, but not completely random either.

Secondly we need to calculate the Area Under Curve value. The code for calculating this value is outlined in algorithm 1, and figure 4.14 demonstrates the calculation of the AUC value of figure 4.13. In this case the AUC value is 0.76, which is in between 0.5 (a random score) and 1.0 (a perfect score).

Algorithm 1 Code to calculate the AUC value.

```
# Loop through a list of 0's and 1's
for mark in rocMarks:
    if mark == 1:
        nTrue = nTrue + 1
    elif mark == 0:
        # In this case, totalPositives = 10 and totalNegatives = 10
        area += (nTrue/ totalPositives) * (1.0/totalNegatives)
```

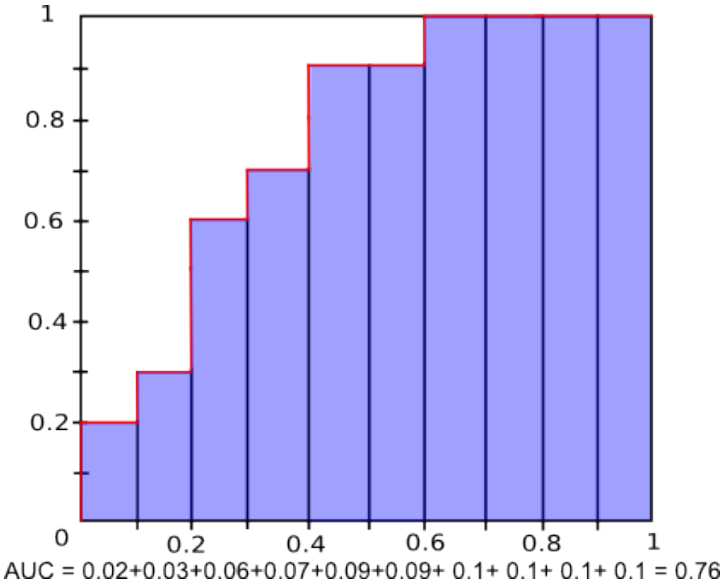

Figure 4.14: Calculating the AUC value of figure 4.13.

4.2.6.3 Function difference comparison statistics

This statistic plot do some simple measurements for benchmarks on base pair probability level. It takes the values returned from the FunctionTrackAnalysisStat and calculates the following measurements:

- **Average Value Within Answer:** Calculates how high the algorithm scores within an answer segment. The higher this measurement is, the better.

- **Average Value Outside Answer:** Calculates how high the algorithm scores outside an answer segment. The lower this measurement is, the better.
- **Difference:** The difference between the above measurements. This should be as high as possible, and is the most important measurement to consider.

These measurements will then be plotted using matplotlib and saved as a png file like the one in figure 4.15, in addition to an html table containing the actual values of the same measurements.

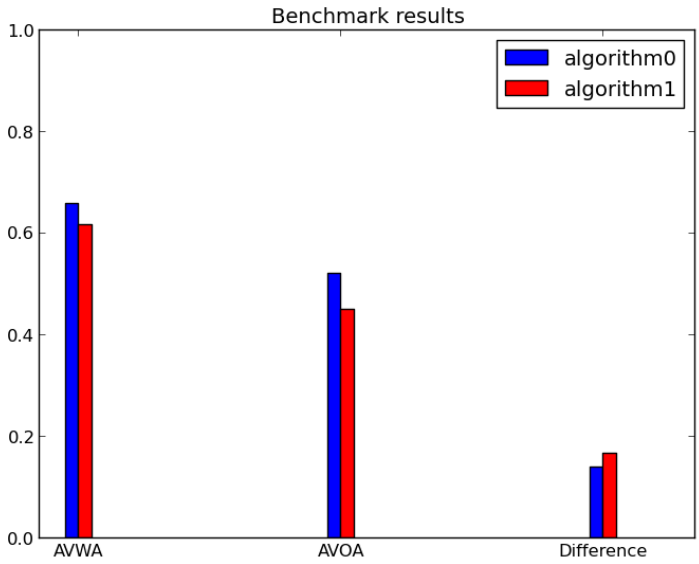

Figure 4.15: Shows the difference between the average within and outside of answer segments.

4.2.6.4 Function score distribution statistic

This statistical plot creates score distribution curves based on results from FunctionTrackScoreDistributionStat. As the number of occurrences for each score is way fewer for the answer segments compared to the entire function (especially in the cases of imbalanced data), we first convert the occurrences to percentages by dividing each element in the array by the total number of scores in the array. Next the two functions will be plotted using matplotlib, as shown in figure 4.16.

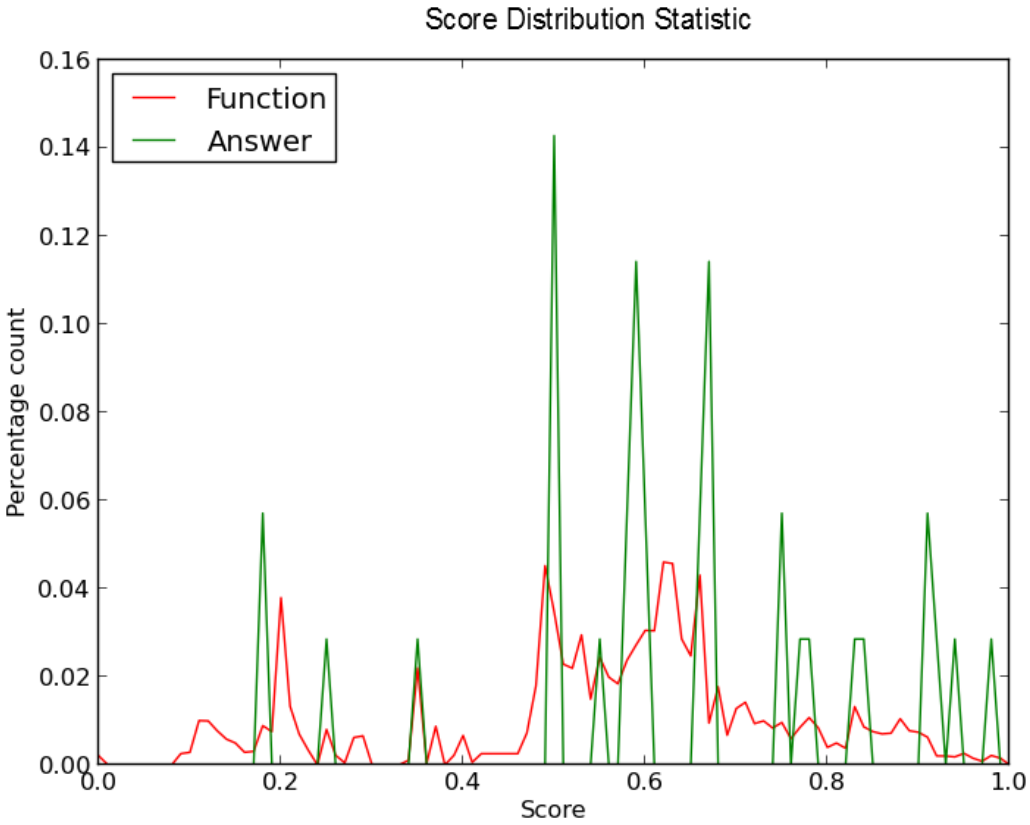

Figure 4.16: Shows the result from a score distribution statistic.

Chapter 5

Results

5.1 Evaluation

In this section we will present the results of the benchmark test sets. As mentioned when we presented these test sets in section A.2, these are not meant to be realistic benchmarks, but rather a demonstration of the benchmark system functionality.

If these were to be realistic benchmarks we would have put more work into optimizing parameters, finding better test sets and more algorithms to test. But that would have been beyond the scope of this thesis as there simply was not enough time to invest into this part. Instead we will comment, and explain how we can interpret the results, and what does these results tell us about the algorithm.

For the motif discovery test sets we ran them on three different algorithms; Weeder, MEME and YMF. All of which was introduced in section 2.3.2.1.

For the gene prediction test set we ran it on Prodigal and Glimmer which was introduced in section 2.3.1.1. Genemark was not included because of limited time.

The base pair probability level test set we ran on one naive machine learning algorithm and a Support Vector Machine (SVM) algorithm.

The Hyperbrowser histories for all these benchmarks can be found in appendix A.2.

5.1.1 Motif discovery sequence level benchmark results

From figure 5.1 we can see from the ROC curve and the statistical measurements that weeder outperforms both algorithm, with YMF comes in second. MEME on the other hand has a correlation coefficient of around zero, which tell us the result is no better than a random result.

Additionally we can see MEME scores high on specificity and zero on sensitivity. This means it manages classify most negative sequences as negative, but no positive sequences as positive. This tells us that MEMEs predictions is rather strict, and found very few positives in a test set. Sometimes this is a desirable trait, and it depends on the test set, which

in this case consists of many short sequences. However in this case, and in the other tests as well we can see that MEME performs poorly because of very strict predictions. This however is something that can be optimized in the parameters, and should be fine tuned for a proper evaluation.

Weeder and YMF on the other hand come up with a high sensitivity and quite low specificity which indicates that they manage to classify most positive sequences correctly, but also marks most negative sequences as positive, meaning the predictions come up with too many positives.

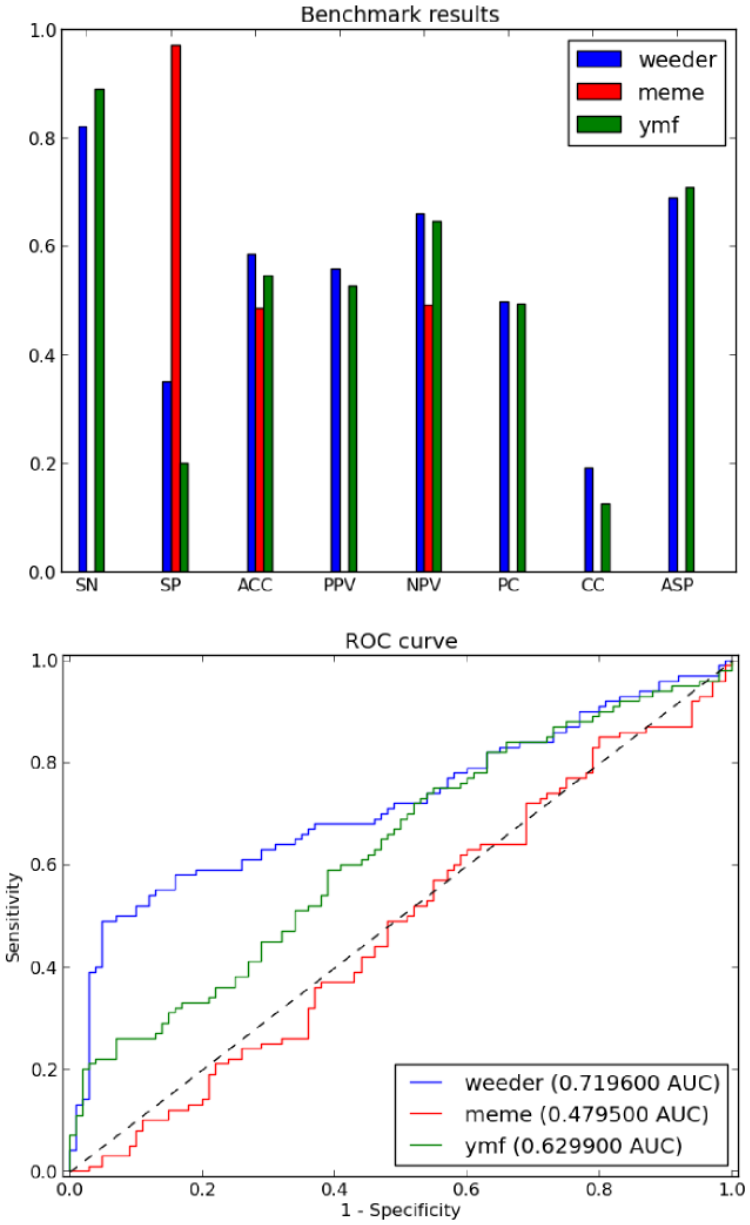

Figure 5.1: The results from the motif discovery benchmark on sequence level.

5.1.2 Motif discovery sequence level benchmark suite results

For the benchmark suite we can see from figure 5.2 the global results across 20 different test sets. In the browser we can also see the results from the individual test sets which is not included here because of space.

The results from the benchmark suite tells us pretty much the same as the above benchmark. MEME does it slightly better this time, but still far behind the other two. Weeder is still the winner, as we can see from the ROC Curve and the correlation coefficient.

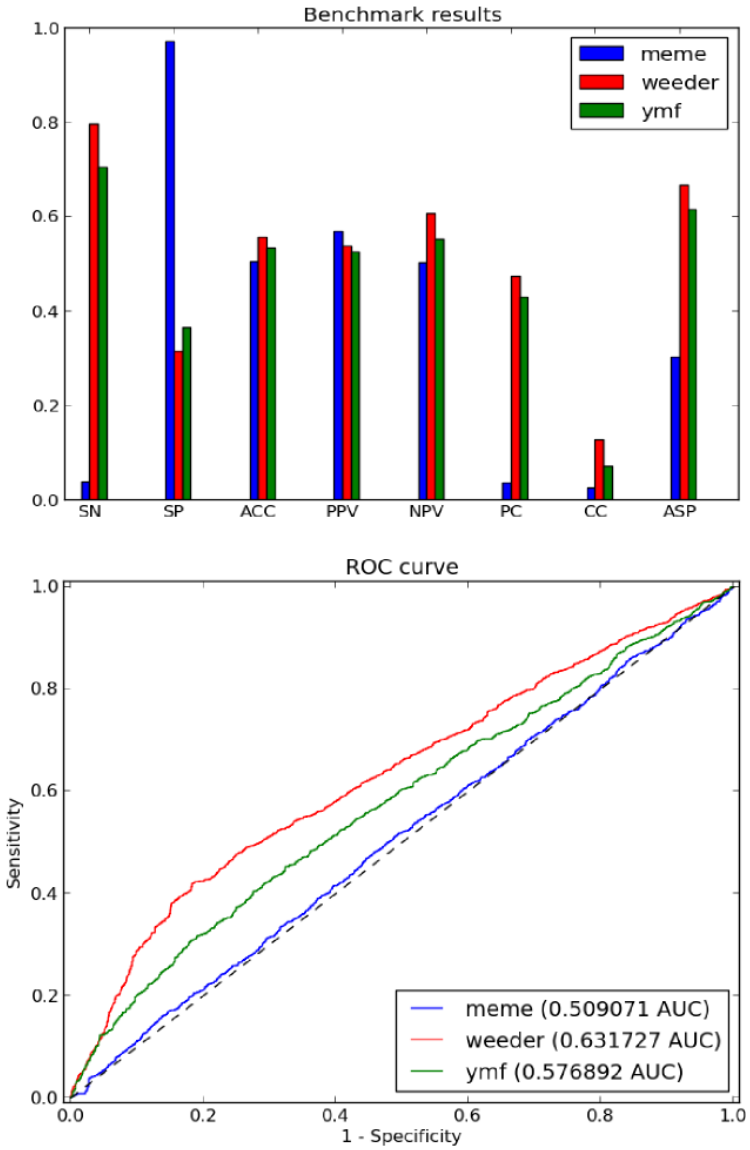

Figure 5.2: The results from the motif discovery benchmark suite on sequence level.

5.1.3 Motif discovery nucleotide level benchmark results

The performance in this test set was rather bad for all algorithms. As we can see from the ROC curve and correlation coefficient, no algorithm come up with a better than random result (actually worse).

As we can see from figure 5.3, all algorithms scores high on specificity and accuracy. This tells us the test set is quite imbalanced with a large number of negatives, which is one of the downside with nucleotide level benchmarks. A high accuracy is therefore a bit deceiving because of all the true negatives.

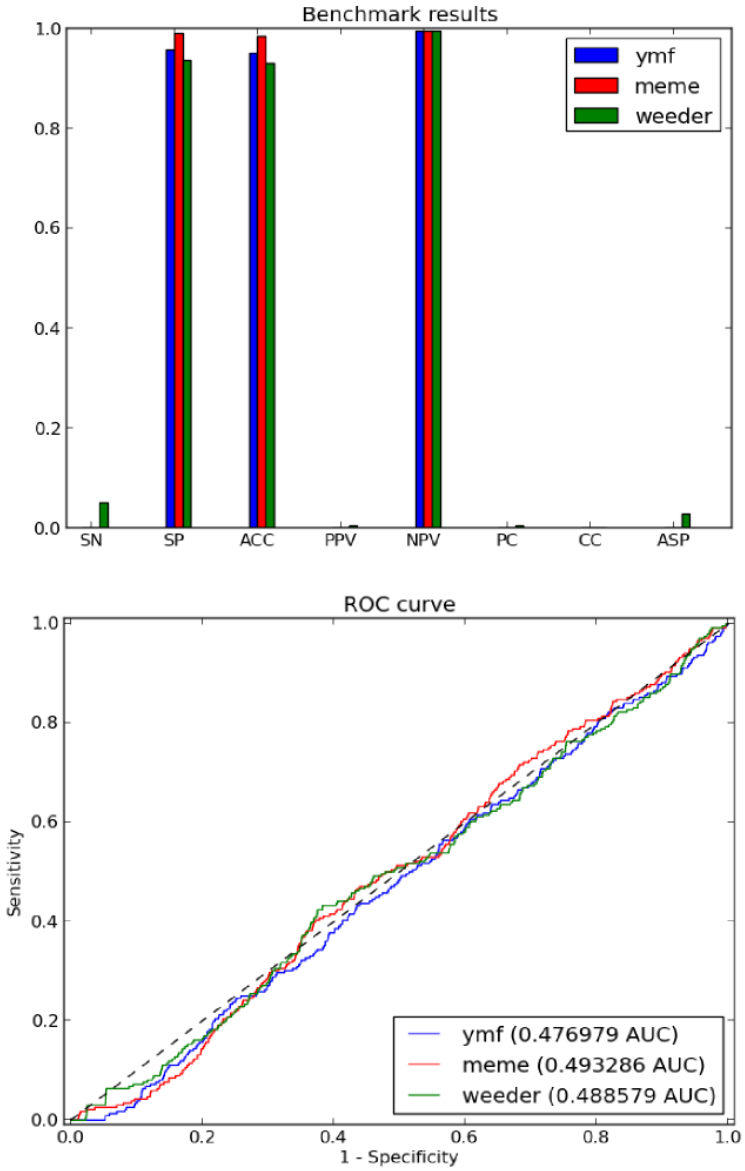

Figure 5.3: The results from the motif discovery benchmark on nucleotide level.

5.1.4 Motif discovery nucleotide level benchmark suite results

For the benchmark suite we can see that Weeder outperformed the other by being the only algorithm with a better than random result. Other than that we get pretty similar results as the above benchmark.

Just like on sequence level, we can click on the results of all the 26 test sets in the browser for more details on individual results.

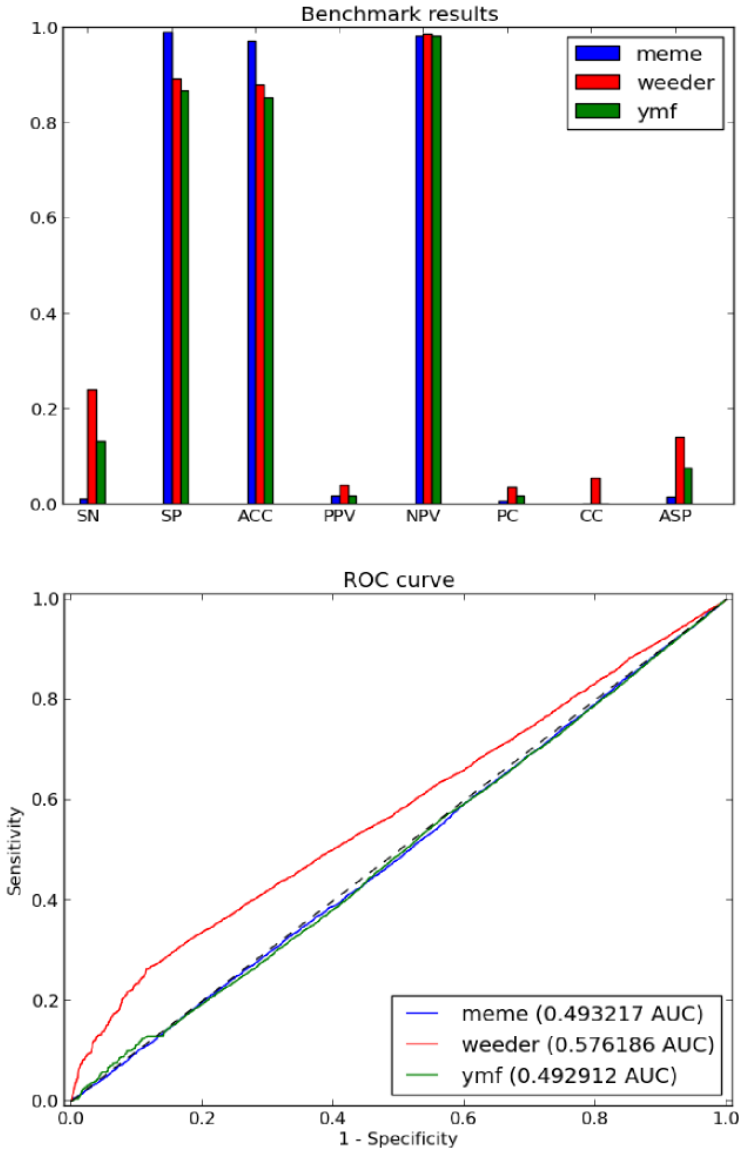

Figure 5.4: The results from the motif discovery benchmark suite on nucleotide level.

5.1.5 Gene prediction nucleotide level benchmark results

As we can see from figure 5.5, Prodigal is clearly outperforming Glimmer. Both algorithm scored quite similar on every measurement, but Prodigal is ahead on every one of them. Additionally, they both score pretty similar on the sensitivity and specificity measurements, meaning that the strictness of the test is just about right.

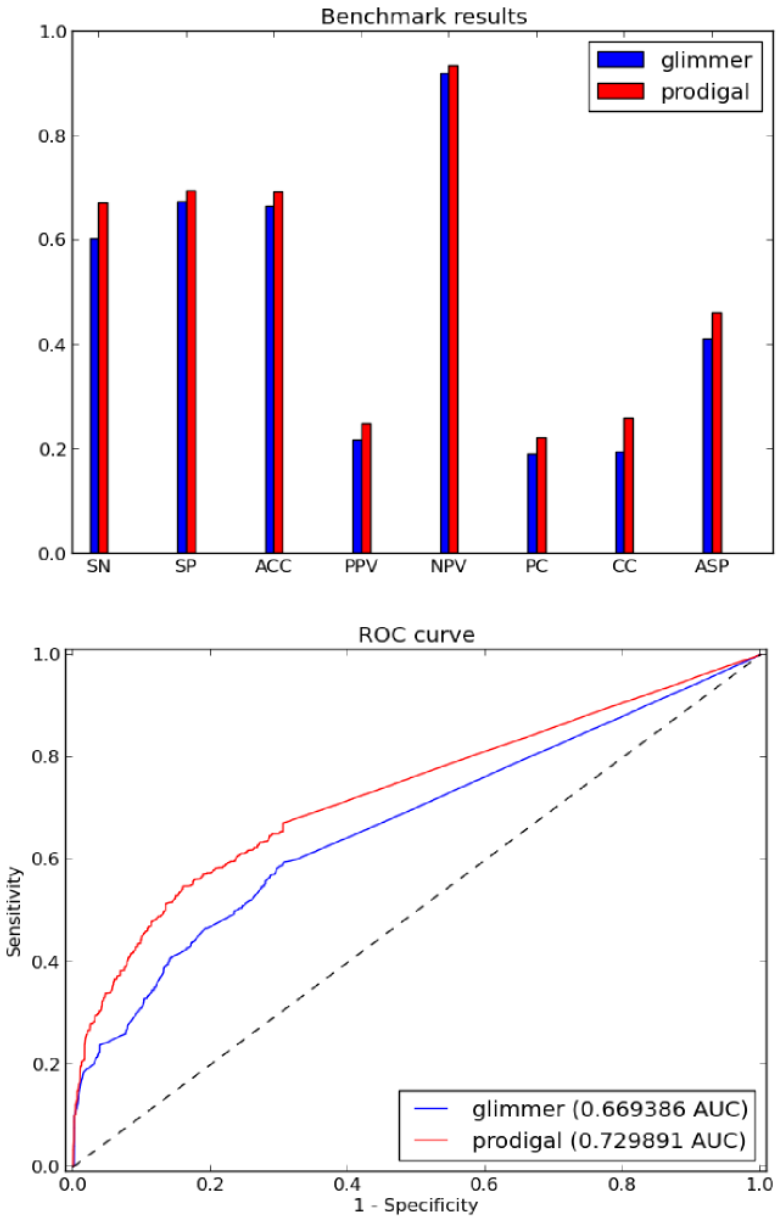

Figure 5.5: The results from the gene prediction benchmark on nucleotide level.

5.1.6 Base pair probability level benchmark results

Algorithm0 is the naive machine learning, and we can see from the difference comparison statistic and the ROC that it performed worse than a random prediction. Algorithm1 on the other hand, which is the SVM algorithm, shows that it managed to predict a higher average score inside the answer. And from the ROC curve we can see that it managed to achieve a better than average prediction.

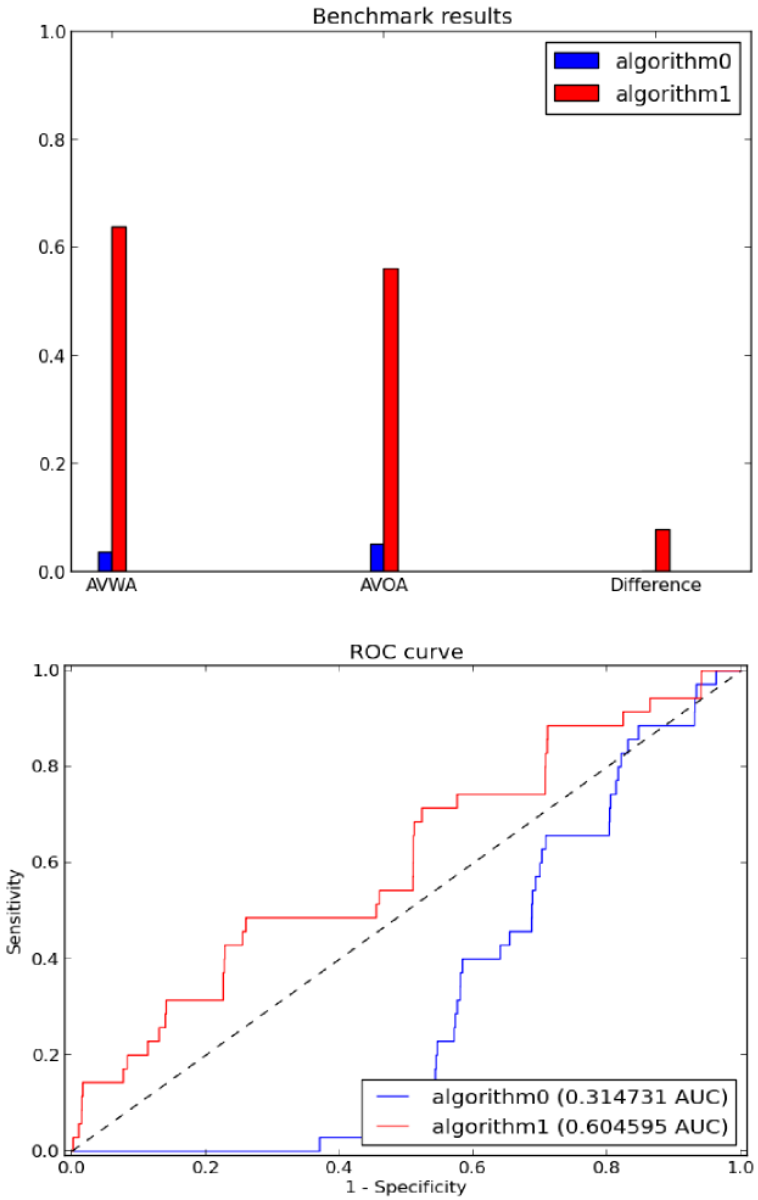

Figure 5.6: The results from the base pair probability level benchmark.

For the score distribution curves we first of all notice that the plots are difficult to evaluate with the naked eye. The first plot is the naive

algorithm, and we can slightly see that the green curve (the answer) receives lower scores than the red curve (rest of the function), though it's hard to tell for sure. The second plot is the SVM algorithm, and we can see that the green function has more peaks at the higher scores, which is a good thing. So from this evaluation we can conclude that the SVM algorithm is a better choice than the naive algorithm.

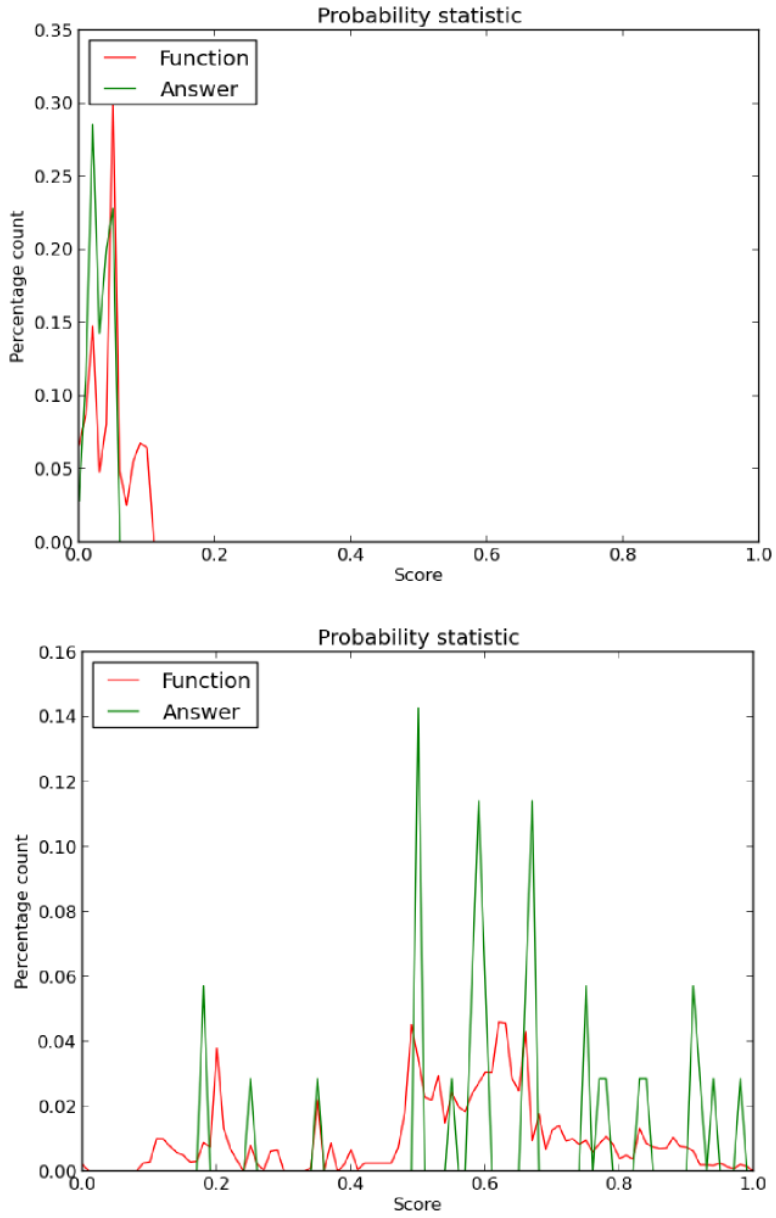

Figure 5.7: The score distribution curve of both algorithms.

Part III

Discussion and Conclusion

Chapter 6

Discussion

In this chapter we will discuss various aspects of the benchmarking system. First we will discuss the results from the previous chapter. Next we will discuss the systems usability from a user standpoint. Finally we will look into some of the challenges we met during the development of this system.

6.1 Results discussion

In the last chapter we can see the results of the test sets we introduced in section A.2. In this section we will discuss what the results tell us, and if they indicate that our system is of any use.

6.1.1 Answers to research questions

In the introduction we posed some research question to reflect the goals we wished to achieve by creating this system. The test sets we created to demonstrate that the system was capable of meeting the demands of these research questions.

1. **Is it possible to streamline the approach of creating and evaluating benchmarks for similar problem areas?** In this thesis our main focus have been to create a streamlined approach to create and evaluate benchmarks for binary classification problems. We created test sets both for gene prediction and motif discovery to demonstrate that the same system can be used to benchmark different problem areas that shared the same input, output and evaluation. These problem areas shared the same functionality without any real challenges, therefore it is safe to assume that the same approach can also be applied to other similar problems such as splice site prediction and nucleosome prediction.

2. **Can this streamlined approach be flexible in both size and type of benchmark, to fit the needs of the user?** We tested the system with both single and multiple test sets, which demonstrated that the system should be able to handle benchmarks of various size. In regard to benchmark type, we tested with two different kinds of

benchmarks for binary classification, nucleotide level and sequence level. Additionally, the evaluation of base pair probability predictions provides a third and different alternative. This should provide the user with some choice of how they wish to define their test sets.

3. **Can the same approach be expanded to fit problem areas that requires different evaluation?** We did have some plans to expand this system to cover other problem areas such as read mapping to prove that the system could be expanded to support different problem areas. However, in the end time was short, and we instead chose to focus on the evaluation of base pair probability predictions to demonstrate the systems ability to change evaluation based on the benchmark specification. However, it's still an open question how the system will be able to handle completely different problem areas in bioinformatics. We will discuss this topic further in section 6.3.2.

6.1.2 Runtime performance

Even though execution time is not of that much importance for a benchmarking system, it is still convenient that the evaluation happens within reasonable time.

The most time consuming benchmark to run was the evaluation of base pair probability predictions, which took around two and a half hour to evaluate each algorithm. The bottleneck here was the preprocessing of tracks, which took about two hours. Preprocessing of tracks is an integrated process in Hyperbrowser that is executed before running a statistic, and it's therefore difficult to optimize this stage. The long evaluation time is of no big surprise however, because of the enormous size of the test set.

The benchmark suites however, which is more realistic in size, took about five minutes each to run with three different algorithms. The biggest bottleneck here was the parsing and conversion of files in GTrackConverter which took about two and a half minute. Five minutes is anyway an acceptable runtime for the evaluation.

6.2 Usability

In this section we will discuss the usability of the system at its current stage. We will look at advantages and disadvantages from using this system, both from a creators and a users perspective.

6.2.1 Benchmark creators point of view

For benchmark creators, the biggest advantage of using this system is that it has all the functionality for evaluation and publishing the benchmark already integrated. Therefore there will be no need for develop any code, which is ideal for biologists with no programming experience.

Additionally, the system is built on the Galaxy framework, which should be familiar to many biologists.

The biggest challenge for the benchmark creator is therefore to create the test sets, which involves specifying the genetic regions to test in the GTrack format. A test set can be created using different approaches, depending on the problem area, and its really up to the benchmark creator which ones he prefers.

Even though GTrack is quite a new and unknown format that was introduced in 2011 [17], its quite similar to other formats like BED and should be easy for biologists to get into. So overall, there can be little doubt that using this system to create a benchmark is easier and less time consuming than building the benchmark from scratch.

6.2.2 Benchmark users point of view

For the benchmark users the difference from using this system over other existing benchmarks is not as significant. They still have to download test sets, and set up and run the test sets on the algorithms of their choice. And even though the system do support a few file formats for motif discovery and gene prediction, the selection of formats is still not very satisfying.

Therefore, the benchmark user still have to find some way of converting most of these result files back to the GTrack format. This first of all requires some understanding of the GTrack format, in addition to creating scripts to convert the result files to GTrack. This is both time consuming and not very user friendly.

On a positive note, these problems already existed with existing benchmarks, and this system do have advantages over these benchmark by addressing some of these issues. But there are still a long way to go to make execution and evaluation of benchmark more user friendly.

6.3 Challenges

Creating a dynamic benchmarking system with our approach is not without its problems. Here we discuss some of the challenges that arose during the development of this system, and some flaws of our approach that might cause trouble for development in the future.

6.3.1 Input and output

One of the bigger challenges we ran into developing this system, and is something that might cause a bit of a challenge for the future, is that our system was based on somewhat naive assumptions about the input and output of the algorithm.

The first challenge is that the input sequences we feed the algorithm in the FASTA format introduced in section 2.5.5, is defined by "our own" description line, specifying the chromosome and start and end positions of the sequence, like this: ">chr1:0-20". However, some algorithms requires

that these description lines follow certain other conventions, making the use of our test sets difficult. In these cases we would therefore first have to change the description lines to a form that the algorithm accepts. But after that comes the challenge of converting the result files back to GTrack. Even though we never tried this, the conversion might cause some trouble because of the loss of info in the description line.

Another problem we faced was that certain result files did not contain the description lines and their relative positions, but only the result sequences. This way we can not use the description line to easily convert the format back to GTrack.

There is a workaround for this problem however, and that is to search for the result sequence in the original test set. Thereby, we can retrieve the position from the description line of the test set and the relative position of the result sequence, but it's not a very elegant approach.

6.3.2 Generalization

As the system is meant to support more problem areas than was implemented during this project, let's discuss the possibilities and challenges that arises if we want to apply our approach to other problem areas. In this section we will look into other relevant problem areas and discuss the possibility of implementing benchmark support for these problems. For a problem area to be applicable for benchmarking with our system, there are certain prerequisites that the problem must follow:

- **Data support:** The first requirement is that we need to able to represent all data (ie. test sets and result files) in a format which can be supported by the Hyperbrowser system.
- **Test set retrieval:** Secondly, one must be able to retrieve the test set from Hyperbrowser in one way or another.
- **Applicable for statistical analysis:** Thirdly, the results should be in format supported by statistical analysis in Hyperbrowser.
- **Evaluation:** Finally, we should be able to display the benchmark results using either html, matplotlib or similar libraries.

Following these prerequisites, we should be able to add benchmark support for a problem area without too much trouble in Hyperbrowser. Going beyond these would require more complex functionality in our system, but that would go beyond what the system was intended to support, and should not be a priority.

6.3.2.1 Read mapping

Read mapping is the process of recovering the placement of sequenced fragments by looking at a very similar reference genome, which should be a representative example of one species genome [43].

Read mapping algorithms takes a set of DNA reads (short DNA sequences) as input in FASTA format. As output they return a set of locations along a reference genome. This is not all that different from the problem areas we've looked at in this thesis, and should be a possible expansion for the system.

6.3.2.2 Sequence assembly

Sequence assembly takes several bits of pieces of DNA data, and tries to put these pieces back together to form the original DNA sequence [30].

Sequence assembly algorithms takes a set of DNA reads as input in FASTA format. And as output they return a set of DNA sequences. Implementing benchmark support for sequence assembly should also be possible using our system. One challenge in the evaluation however, is that the statistical analysis in Hyperbrowser is meant to compare tracks and not sequences, which might cause some trouble in the implementation.

6.3.2.3 Multiple sequence alignment

The problem of Multiple Sequence Alignment is to discover similarities in three or more DNA, RNA or protein sequences, in order to find a shared evolutionary origin between different species [39].

Multiple sequence alignment algorithms takes a set of either DNA, RNA or amino acid sequences in FASTA format as input, and returns an alignment of all the different sequences. Representing these alignments in a format like GTrack, and doing statistical analysis on this data can prove to be difficult, as the statistical analysis is focused on comparing two tracks and not alignments between several.

Additionally, most modern benchmarks for multiple sequence alignment incorporate 3D structural information in their evaluation, which will be another challenge to implement. And with many excellent existing benchmarks like BALIBASE, [2] we see no need to implement this as part of the system.

6.3.2.4 Protein structure prediction

Protein structure prediction is the process of predicting the three-dimensional structure of a protein, given a sequence of amino acids [25].

Protein structure prediction algorithms takes a sequence of amino acids as input in FASTA format, and returns a 3D representation of a protein. As Hyperbrowser was never meant to handle 3D representations of proteins, we believe the implementation will be hard without adding a lot of new features. Therefore, we would advice using other existing benchmarks for protein structure prediction instead.

6.3.3 Architecture

With more problem areas and more features comes more complexity. Therefore it's important that the underlying architecture of the system is

solid to facilitate new additions without creating one big mess. In this section we will discuss the current architecture, its weaknesses and suggest some improvements.

6.3.3.1 Shared service class

The first architectural flaw to point out is the class "BenchmarkUtil", which serves as a common service class for all the tools. This class seemed like a good idea at the time to avoid duplication of code, but looking back at it the idea of having one shared service class for different tools turned out to be a bad idea.

The reason for this is first of all the sheer size and different features that will be put into this class as the system gets bigger and more complex. This will make it hard to keep a clear overview of the system, and know what function is called from what class. This will in the end make the system difficult to maintain consistent, as one change in this class might cause trouble in three different other classes.

6.3.3.2 Lack of testing

Another weakness with our approach was to develop this system without using test driven development. The idea of test driven development is to first create a set of tests that tests all the functionality of every function, before the actual implementation. The advantage of this approach is that we are forced to put more effort into thinking about the design and functionality of the system before the implementation.

This will make us less likely of doing major design flaws that will be changed afterwards. In this project we made several changes to the architecture underway, that could have surely been avoided if we had put more effort into the design stage. Additionally it will also be easier to make the system consistent and avoid unforeseen bugs, as the system can easily be tested for every change we make. Currently, the only way to test our system is to manually run all the test sets from creation to evaluation, which is a very inefficient way of testing the application.

6.3.4 Interpretation of score distribution curves

One of the new evaluation methods we introduced in this thesis was the score distribution curves in section 2.4.2.2 for evaluating base pair probability predictions. The purpose of this plot was to compare the score distribution of the answer scores with the scores of the entire function.

However as we can see from the results in section 5.1.6, it's hard to compare the two function and conclude which one is better than the other. Some of the reason for this is because of the imbalanced data in the test set. With only 35 positives out of 48 million, we get more peaks than a function. However, we do suspect that this is not a very viable method for evaluating predictions, but can rather be a useful tool to see the score distribution of a prediction.

6.3.5 Genome availability in the Hyperbrowser system

A final challenge we will discuss is that Hyperbrowser only come with a limited selection of genomes integrated in its system. This means that it won't be possible to create test sets with other genomes than the ones that Hyperbrowser offers.

Hyperbrowser now has roughly 150 genomes as part of its system, so it does provide some selection. Yet there are still many genomes missing, especially when it comes to prokaryotes, which might make the benchmark system less useful for some.

Chapter 7

Conclusion and future work

In this chapter we will wrap up our work by first looking at what work remains, and then conclude our project by giving some final thoughts and reflections.

7.1 Future Work

The dynamic benchmarking system was meant to demonstrate that we can use a common system to benchmark different problems. Even though we implemented many core features, there are still a lot of work remaining before its full potential can be realized. Here we will look into what additions that can be made to make the system more useful in the future.

7.1.1 Integrated algorithms

As the system is right now, the user have to download test sets and set up the algorithms themselves. This can be a time consuming process, especially if documentation is lacking and there are many algorithms to test.

A useful addition to the benchmark system would therefore be to have some of the more popular algorithms already integrated into the system. This way the test set can be ran on a selection of different algorithms with just a push of a button.

Though useful, this needs some care in the implementation. To be done right, we should be able to set and optimize each parameter just like we would do if we ran the algorithm from the command line.

7.1.2 More file formats

As the system is aimed at biologists with little programming knowledge, manually converting result files from their original format to GTrack can be somewhat of a challenge. Therefore its vital that the system do support the output from the most used algorithms for each problem area it supports.

Currently the benchmark system support the output from six different algorithms for motif discovery and gene prediction. This is a start, but far

from good enough if someone want to use the system to set up a proper benchmark.

A job for the future will therefore to look at what file formats exists, and convert these to GTrack, and add the code to GTrackConverter.

7.1.3 More problem areas

In this thesis our only focus was to support binary classification problem areas. But there exists many other problem areas in bioinformatics that can take advantage of this system, for instance read mapping and sequence assembly.

When adding support for more problem areas, there are a few things one must consider:

- **Benchmark specification:** What will the benchmark specification include. How will the test sets look like etc.
- **Retrieving test sets:** How will we retrieve the test set from hyperbrowser.
- **Evaluation:** What evaluation methods exist for this problem area, and which ones to implement.
- **Documentation:** Finally its important that new problem areas are properly documented in the tutorials.

7.1.4 Test set generation

Currently the benchmark creator have to manually create the GTrack files for the test set. This can be, depending on the scale and the approach he uses, a time consuming and daunting task.

One improvement in the future will therefore be to provide some tool to help define and generate the GTrack files for the test sets. Even though its hard to describe exactly what this tool might look like, Hyperbrowser already comes with a lot of functionality that can come in handy when developing such a tool.

This might be key for the success of the system, as simplifying the creation of test sets will be an incentive for researchers to create relevant test sets for each problem area that can be shared with the community.

7.1.5 Better evaluation

Even though we implemented some popular evaluation methods such as ROC curves and included some of the most used measurements, there are more methods out there that can be implemented as well.

For instance Precision Recall (PR) curves which was introduced in section2.4.1.3 is commonly used in the evaluation of binary classification problems, but was not implemented during this project.

Additionally, the results as they stands right now does not give a very clear picture about what algorithm gives the best result, unless the user

is already familiar with the evaluation methods. A nice addition would therefore be to give some extra explanation of the results, and maybe give the results an overall ranking based on the measurements and ROC results.

7.1.6 Improve the architecture

An important task for the future is to approve the architecture of the current system, as discussed in section 6.3.3. This includes creating separate service classes for all the three tools, and make the system test driven by creating tests for existing functionality.

7.2 Conclusion

In this thesis we have looked at some of the problems with benchmarking in bioinformatics today. Then we proposed a new benchmarking platform to address some of these problems. With this system we wanted to offer a common platform for biologists and bioinformaticians to create, share benchmarks across different problem areas.

Though the system offers benchmark support for a few problem areas, it still has a long way to go to before it can serve as a complete benchmarking platform. Its success therefore relies on the community to create test sets, and expand on its functionality.

The goal of this project, however, was to provide a proof of concept of how we can create benchmarks in the future. In that respect, I believe we've succeeded.

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990.

[2] M. R. R. Aniba, O. Poch, and J. D. Thompson. Issues in bioinformatics benchmarking: the case study of multiple sequence alignment. *Nucleic Acids Research*, 38(21):7353–7363, November 2010.

[3] A Annunziato. Dna packaging: Nucleosomes and chromatin. *Nature Education*, 1(1):10–9, 2008.

[4] A. Arvey, P. Agius, W. S. Noble, and C. Leslie. Sequence and chromatin determinants of cell-type-specific transcription factor binding. *Genome Research*, 22(9):1723–1734, September 2012.

[5] T. L. Bailey, N. Williams, C. Misleh, and W. W. Li. Meme: discovering and analyzing dna and protein sequence motifs. *Nucleic Acids Research*, 34(suppl 2):W369–W373, July 2006.

[6] M. Borodovsky and D. Mcininch. Genemark: Parallel gene recognition for both dna strands. *Computers & Chemistry*, 17:123–133, 1993.

[7] M. Burset and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, 34(3):356–357, June 1996.

[8] S Clancy. Rna splicing: introns, exons and spliceosome. *Nature Education*, 1(1):1, 2008.

[9] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM.

[10] Numpy developers. Numpy official website. Retrieved April 26, 2013, from http://www.numpy.org.

[11] L. Elnitski, H. Piontkivska, and L.R. Welch. *Advances in genomic sequence analysis and pattern discovery*. Science, engineering, and biology informatics. World Scientific Publishing Company, Incorporated, 2011.

[12] S. Engebretsen. Evaluation of gene prediction methods for prokaryotes. Master's thesis, Institute for Informatics, University of Oslo, 2012.

[13] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006.

[14] Python Software Foundation. Python programming language - official website. Retrieved April 26, 2013, from http://www.python.org/.

[15] Python Software Foundation. Python regular expressions. Retrieved April 26, 2013, from http://docs.python.org/2/library/re.html.

[16] J. Goecks, A. Nekrutenko, J. Taylor, and Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86+, August 2010.

[17] S. Gundersen, M. Kalas, O. Abul, A. Frigessi, E. Hovig, and G. Sandve. Identifying elemental genomic track types and representing them uniformly. *BMC Bioinformatics*, 12(1):494+, December 2011.

[18] F Haaland. Using machine learning to make sense of our genome. Master's thesis, Institute for Informatics, University of Oslo, 2013.

[19] H. He and E. A. Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, September 2009.

[20] M. Holtgrewe, A. K. Emde, D. Weese, and K. Reinert. A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, 12(1):210+, 2011.

[21] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[22] D. Hyatt, G. L. Chen, P. F. Locascio, M. L. Land, F. W. Larimer, and L. J. Hauser. Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics*, 11(1):119+, 2010.

[23] A. V. Lukashin and M. Borodovsky. Genemark.hmm: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–15, 1998.

[24] C. Mathé, M. F. Sagot, T. Schiex, and P. Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids Research*, 30(19):4103–4117, October 2002.

[25] J. Moult. A decade of casp: progress, bottlenecks and prognosis in protein structure prediction. *Current opinion in structural biology*, 15(3):285–289, June 2005.

[26] NCBI. A basic introduction to the science underlying ncbi resources. Retrieved April 26, 2013, from http://www.ncbi.nlm.nih.gov/About/primer/genetics_genome.html.

[27] G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in dna sequences. *Bioinformatics (Oxford, England)*, 17 Suppl 1(suppl 1):S207–S214, June 2001.

[28] G. Pavesi, P. Mereghetti, G. Mauri, and G. Pesole. Weeder web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. *Nucleic Acids Research*, 32(suppl 2):W199–W203, July 2004.

[29] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, April 1988.

[30] M. Pop, S. L. Salzberg, and M. Shumway. Genome sequence assembly: algorithms and issues. *Computer*, 35(7):47–54, July 2002.

[31] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated markov models. *Nucleic Acids Research*, 26(2):544–548, January 1998.

[32] G. Sandve, S. Gundersen, H. Rydbeck, I. Glad, L. Holden, M. Holden, K. Liestol, T. Clancy, E. Ferkingstad, M. Johansen, V. Nygaard, E. Tostesen, A. Frigessi, and E. Hovig. The genomic hyperbrowser: inferential genomics at the sequence level. *Genome Biology*, 11(12):R121+, December 2010.

[33] G. K. Sandve, O. Abul, V. Walseng, and F. Drablos. Improved benchmarks for computational motif discovery. *BMC Bioinformatics*, 8(1):193, 2007.

[34] G. K. Sandve and F. Drablos. A survey of motif discovery methods in an integrated framework. *Biology direct*, 1(1):11+, April 2006.

[35] J. Sapp. The prokaryote-eukaryote dichotomy: meanings and mythology. *Microbiology and molecular biology reviews*, 69(2):292–305, 2005.

[36] S. Sinha and M. Tompa. A statistical method for finding transcription factor binding sites. *Proceedings / ... International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology*, 8:344–354, 2000.

[37] S. Sinha and M. Tompa. Ymf: A program for discovery of novel transcription factor binding sites by statistical overrepresentation. *Nucleic Acids Research*, 31(13):3586–8, 2003.

[38] Y. Tanaka and K. Nakai. An assessment of prediction algorithms for nucleosome positioning. *Genome Inform*, 23(1):169–78, 2009.

[39] J. D. Thompson, P. Koehl, R. Ripp, and O. Poch. Balibase 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins*, 61(1):127–36+, 2005.

[40] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. *Nature biotechnology*, 23(1):137–44, 2005.

[41] Z. Wang, Y. Chen, and Y. Li. A brief review of computational gene prediction methods. *Genomics, proteomics & bioinformatics / Beijing Genomics Institute*, 2(4):216–221, November 2004.

[42] E. Wingender, X. Chen, R. Hehl, H. Karas, I. Liebich, V. Matys, T. Meinhardt, M. Pruss, I. Reuter, and F. Schacherer. Transfac: an integrated system for gene expression regulation. *Nucleic Acids Research*, 28(1):316–319, 2000.

[43] V. Yanovsky, S. M. Rumble, and M. Brudno. Read mapping algorithms for single molecule sequencing data. In *Proceedings of the 8th international workshop on Algorithms in Bioinformatics*, WABI '08, pages 38–49, Berlin, Heidelberg, 2008. Springer-Verlag.

[44] H. Yao, L. Guo, Y. Fu, L. A. Borsuk, T. J. Wen, D. S. Skibbe, X. Cui, B. E. Scheffler, J. Cao, S. J. Emrich, D. A. Ashlock, and P. S. Schnable. Evaluation of five ab initio gene prediction programs for the discovery of maize genes. *Plant molecular biology*, 57(3):445–460, February 2005.

[45] L. Zhang and L. Luo. Splice site prediction with quadratic discriminant analysis using diversity measure. *Nucleic Acids Research*, 31(21):6214–6220, 2003.

Appendix A

-

A.1 The benchmark system

The benchmark system implemented in this project is available here:

- http://hyperbrowser.uio.no/bench/

The tools implemented in this thesis are; Benchmark Creation Tool, Benchmark Retrieval Tool and Benchmark Evaluation tool. These tools are available under "Restricted and experimental tools" at the left pane.

A.2 Hyperbrowser benchmark histories

The Hyperbrowser histories for all the benchmarks are available on the web. Here, all the test sets, results and tools executed in this thesis are available for anyone to see. The results from each tool can be viewed by clicking "the eye". Important! In order to view the results from the benchmark suites, scroll all the way down to the bottom of the page. The histories can be found here:

- http://hyperbrowser.uio.no/bench/u/anderrb/h/motif-discovery-benchmark-on-sequence-level-2
- http://hyperbrowser.uio.no/bench/u/anderrb/h/motif-discovery-benchmark-suite-sequence-level-1
- http://hyperbrowser.uio.no/bench/u/anderrb/h/motif-discovery-benchmark-nucleotide-level-1
- http://hyperbrowser.uio.no/bench/u/anderrb/h/motif-discovery-benchmark-suite-nucleotide-level-2
- http://hyperbrowser.uio.no/bench/u/anderrb/h/gene-prediction-benchmark-1
- http://hyperbrowser.uio.no/bench/u/anderrb/h/fredrik-bench

A.3 Test sets and result files

All test sets, results and other files used in the process can be found here:

- http://hyperbrowser.uio.no/dev2/static/downloads/testsets.tar.gz

A.4 Tutorials

We created two tutorials for how to use the benchmark system on the wiki, one for the benchmark creator and one for the benchmark user. These tutorials can be found here:

- http://bmimaster.wiki.ifi.uio.no/Benchmark_Creator_Tutorial
- http://bmimaster.wiki.ifi.uio.no/Benchmark_User_Tutorial

A.5 Code

The code developed in this project is available here:

- http://hyperbrowser.uio.no/dev2/static/downloads/code.tar.gz

The entire branch can be checked out either from the invitro server, or from git hub:

- svn co svn+ssh://$USER@invitro.titan.uio.no/projects/bioinfoprojects/svn_repository/new_hb/branches/bench
- git clone git://github.com/Anderrb/Dynamic-benchmark.git